

April 2015

# NVIDIA Bug Services: Synchronization & Statistics

Joseph Richard Spicola  
*Worcester Polytechnic Institute*

Kristen Lee Brann  
*Worcester Polytechnic Institute*

Tushar Narayan  
*Worcester Polytechnic Institute*

Follow this and additional works at: <https://digitalcommons.wpi.edu/mqp-all>

---

## Repository Citation

Spicola, J. R., Brann, K. L., & Narayan, T. (2015). *NVIDIA Bug Services: Synchronization & Statistics*. Retrieved from <https://digitalcommons.wpi.edu/mqp-all/2569>

This Unrestricted is brought to you for free and open access by the Major Qualifying Projects at Digital WPI. It has been accepted for inclusion in Major Qualifying Projects (All Years) by an authorized administrator of Digital WPI. For more information, please contact [digitalwpi@wpi.edu](mailto:digitalwpi@wpi.edu).

# NVIDIA Bug Services: Synchronization & Statistics

Kristen Brann, Tushar Narayan, Joseph Spicola

March 5, 2015

A Major Qualifying Project Report  
submitted to the Faculty  
of the  
WORCESTER POLYTECHNIC INSTITUTE  
in partial fulfillment of the requirements for the  
Degree of Bachelor of Science  
by:

---

Kristen Brann

---

Tushar Narayan

---

Joseph Spicola

Date: March 2015

Approved:

---

Professor David Finkel, Advisor

This report represents the work of one or more WPI undergraduate students.  
Submitted to the faculty as evidence of completion of a degree requirement.  
WPI routinely publishes these reports on its web site without editorial or peer review.

## Acknowledgements

We would like to thank the following individuals for their help and guidance during this project.

Professor David Finkel, WPI

Mrs. DeAnn Finkel, WPI

Hari Mudaliar, NVIDIA

Andrew Chew, NVIDIA

Christopher Freeman, NVIDIA

Karthik Gopalakrishnan, NVIDIA

Brian Ty, NVIDIA

Lawrence Robinson, NVIDIA

Sumanta Chakraborty, NVIDIA

Silke Steurich, NVIDIA

Wu-Hsuan Lee, NVIDIA

Kuang Heng Li, NVIDIA

Leandro Awa, NVIDIA

Matthew Jansen, NVIDIA

Eric Busto, NVIDIA

Erik Welch, NVIDIA

# Contents

Acknowledgements.....	2
List of Figures .....	6
List of Tables .....	6
List of Code Fragments .....	7
1 Abstract.....	8
2 Introduction .....	9
2.1 NVIDIA.....	9
2.2 Partnership with Google .....	9
2.3 Problem Statement.....	10
2.3.1 Bug Synchronization Service .....	11
2.3.2 Bug Statistics Tool .....	12
3 Background .....	14
3.1 Bug Trackers.....	14
3.1.1 NVIDIA Bug Tracking .....	14
3.1.2 Google Partner Bugs .....	17
3.2 Email Protocols.....	20
3.2.1 SMTP .....	20
3.2.2 POP3.....	21
3.2.3 IMAP.....	21
3.3 Web Scraping .....	21
3.3.1 HTML.....	22
3.3.2 Google Project Hosting HTML Structure.....	22
3.3.3 Retrieving and Parsing HTML.....	23
3.4 Automation .....	23

3.4.1	Daemon.....	23
3.5	Security .....	25
3.5.1	Encryption.....	25
3.6	API Protocols .....	26
3.6.1	XML .....	26
3.6.2	SOAP.....	27
3.6.3	WSDL.....	29
3.6.4	Unicode.....	29
3.7	Logging .....	30
3.7.1	The Python <i>logging</i> Module .....	30
3.7.2	Event Levels .....	30
3.7.3	The YAML Data Format.....	31
3.8	Statistics .....	32
3.8.1	ChromeOS Statistics.....	32
3.8.2	Generating Dynamic Charts with Microsoft Excel .....	35
4	Methodology.....	37
4.1	Integration with Google .....	37
4.1.1	Email Scraping.....	37
4.1.2	Web Scraping.....	40
4.1.3	Comparing Strategies.....	42
4.2	Integration with NVIDIA.....	43
4.2.1	Connecting to NVBugs .....	43
4.2.2	Using the Internal SOAP API .....	44
4.2.3	Using the Partner Bugs API .....	47
4.2.4	API Choice .....	49

4.2.5	Tradeoffs around new Mapping Schema.....	52
4.3	Logging .....	54
4.3.1	The Logger Hierarchy .....	54
4.3.2	Classes and Objects.....	55
4.3.3	Configuration .....	57
4.3.4	Logging Tradeoffs.....	59
4.4	Deployment.....	61
4.4.1	Development .....	61
4.4.2	Testing.....	61
4.4.3	Deploying to Production .....	62
4.5	Statistics .....	63
4.5.1	Preparing the Environment .....	63
4.5.2	Connecting to the Gremlins Database .....	63
4.5.3	Automating SQL Connection with Visual Basic for Applications .....	64
4.5.4	Automating SQL Generation with Visual Basic for Applications.....	65
4.5.5	Creating a User Interface for the Project Managers.....	67
5	Results.....	70
5.1	Bug Synchronization Service .....	70
5.1.1	Extending the Bug Synchronization Service.....	71
5.2	Bug Statistics Tool .....	72
5.2.1	Extending the Bug Statistics Tool.....	72
6	Conclusion.....	73
7	Bibliography .....	74
8	Appendices.....	81
Appendix A	Web Service Definition Language Example .....	81

Appendix B	Example of NVBugs Embedded Charts .....	85
Appendix C	Information Retrieved from a Google Bug .....	86
Appendix D	Example WSBug XML Object .....	87
Appendix E	Bug Synchronization Architecture Overview .....	91
Appendix F	Configuration File .....	93
Appendix G	Deployment Instructions for Google Chrome Sync .....	95
Appendix H	Documentation .....	101
Appendix I	Bug Synchronization Statistics .....	109

## List of Figures

Figure 1: “NVIDIA ChromeOS Bugs” Custom View on NVBugs .....	16
Figure 2: Project Bug Velocity Chart .....	33
Figure 3: Project Bug Trend by Modules .....	34
Figure 4: Project Bug Trend by Engineer .....	34
Figure 5: Example WSBug XML Object .....	45
Figure 6: Sample mapping of NVIDIA Bug to Google Bug using <i>Synopsis</i> field .....	50
Figure 7: Customer Information Section for an NVIDIA Bug .....	54
Figure 8: The User Interface of the Statistics Tool .....	68
Figure 9: UI for Generated Charts .....	69
Figure 10: Embedded Charts on NVBugs .....	85
Figure 11: Bug Synchronization Service Flow Diagram .....	91

## List of Tables

Table 1: Predefined Logging Levels in Python's <i>logging</i> Module .....	31
Table 2: Bug Synchronization Performance Statistics .....	70
Table 3: Bug Synchronization Service Statistics .....	109

## List of Code Fragments

Code Fragment 1: The Paragraph HTML Element .....	22
Code Fragment 2: HTML Fragment from Google Project Hosting.....	22
Code Fragment 3: Sample XML Snippet .....	27
Code Fragment 4: Sample SOAP Message Skeleton.....	28
Code Fragment 5: Sample YAML File .....	31
Code Fragment 6: NVIDIA Partner Bug Audit Trail Notes .....	48
Code Fragment 7: Obtaining a logger object.....	55
Code Fragment 8: Configuration of Logging Formatter.....	57
Code Fragment 9: Sample Log Message .....	57
Code Fragment 10: Setup for a logger object.....	58
Code Fragment 11: Connection Code .....	64

## **1 Abstract**

The NVIDIA ChromeOS team works with NVIDIA's internal bug database and Google's partner bugs database. Project Managers currently manually copy-paste bugs from the Google database to the NVIDIA database so that NVIDIA engineers can work on those bugs. To track the progress of projects, NVIDIA's Project Managers manually tabulate data and generate bug trend visualizations. Both these manual operations are time-intensive and prone to human error.

We developed a service to automate the synchronization of Google bugs, guaranteeing timely and automatic Google partner bug updates. We also developed a tool to automate the generation of bug statistics, providing an easy manner of periodically visualizing bug trends.

## **2 Introduction**

### **2.1 NVIDIA**

NVIDIA is a Visual Computing Company that was founded in 1993. In 1999, NVIDIA invented the graphics processing unit (GPU), which was a significant stride forward for the field of computing graphics. NVIDIA targets markets including gaming, professional data visualization, scientific computing, and general high performance visual computing. NVIDIA invests in cloud services, notably the NVIDIA GRID cloud gaming technology that allows users to stream games [56].

NVIDIA has also entered market sectors that involve smart devices, and provides processor chips for various device types:

- tablets: for example, the Google Nexus 9 [27]
- laptops: for example, the HP Chromebook [36]
- smartphones: for example, the HTC OneX+ [41]

NVIDIA's current generation of mobile processor chip is named the NVIDIA Tegra X1 [58].

NVIDIA has defined a new market segment by releasing the Shield Portable [60], which is a crossover between a handheld gaming console and a tablet computer. In recent months, NVIDIA has also made strides in developing a Visual Computing Module (VCM) that is meant to integrate the NVIDIA Tegra processor into vehicles [62].

Serving these different market segments means that NVIDIA has to work with different partners within each particular market. NVIDIA's partners include Audi [14], Google [26], Microsoft [47], Tesla [88], and many others [57].

### **2.2 Partnership with Google**

In 2014, NVIDIA announced a partnership with Acer, to power their new Google Chromebooks with the NVIDIA Tegra K1 mobile processor [11]. Later that year, HP announced that it would also launch a Chromebook powered by the Tegra K1 [85]. Integrating the Tegra K1 into the Chromebooks dramatically increased performance in battery life, processing speed, and graphics power [11]. NVIDIA's Tegra K1 processor now powers many Google Chromebooks made by Acer and HP:

- Acer Chromebook 13 CB5-311-T1UU [5]
- Acer Chromebook 13 CB5-311-T7NN [1]
- Acer Chromebook 13 CB5-311-T9B0 [6]

- Acer Chromebook 13 CB5-311-T677 [7]
- Acer Chromebook 13 CB5-311-T5BD [8]
- Acer Chromebook 13 CB5-311-T677 [3]
- Acer Chromebook 13 CB5-311-T5XO [4]
- Acer Chromebook 13 CB5-311-T9Y2 [2]
- Acer Chromebook 13 CB5-311P-T9AB [9]
- HP Chromebook – 14-x050nr Touch [40]
- HP Chromebook – 14-x040nr [39]
- HP Chromebook – 14-x030nr [38]
- HP Chromebook – 14-x010nr [37]

Chromebooks run Google's Chrome Operating System (for more information on the Chrome Operating System, see [33]). Due to this relationship, NVIDIA has a team devoted to working specifically with mobile products that run ChromeOS. The ChromeOS team works closely with Google to ensure that the devices perform well with the Tegra K1 processor. This partnership often involves both Google and NVIDIA developers investigating and working on the same bugs.

## 2.3 Problem Statement

Project Managers (PMs) are employees whose job revolves around making sure that their company's products launch on schedule. While the role of a PM can vary broadly depending on the company, PMs are generally responsible for translating company strategy into consumable products and services. PMs work with various stakeholders, including management, engineers, partners, and customers to make sure that projects are delivered according to their deadlines.

PMs use a variety of statistics to track the progress of a project. One of the metrics the PMs rely on is the bugs related to a project. Tracking bugs has a dual purpose:

- Engineers record the progress they are making on fixing a particular issue or developing new functionality on the bug. Bugs can thus provide a timeline of the issues encountered, the steps taken to fix them, and the overall outcome.
- The overall number of bugs per time can provide a view of the stability of the project. The change in bug numbers can be used to gain insight into engineer productivity,

effectiveness of developer code fixes, and the velocity with which engineers are verifying code fixes.

NVIDIA partners with other companies to maintain its market position in supplying graphical processor chips for various device categories (Section 2.1). PMs therefore have an interest in both the bugs in the NVIDIA bug database and the bugs in the project partner's bug database. For NVIDIA engineers to work on bugs that project partners find, the bugs must be copied over from the project partner's database into the NVIDIA database.

Different teams at NVIDIA have different processes in place for copying over bugs from a partner's database to the NVIDIA database. We will be working for the ChromeOS team (Section 2.2) at NVIDIA. The ChromeOS team does not have any automated process in place for copying over partner bugs to the NVBugs database. Consequently, ChromeOS Project Managers spend a lot of time doing manual data synchronization in order to make sure that the NVIDIA bug database is up to date with a partner's bug database.

ChromeOS Project Managers also spend quite some time doing manual data tabulation to generate the charts and trends that provide them insight into project development. Manual data manipulation is time-intensive and prone to error.

We have two solutions for this problem: an automated bug synchronization service, and a bug statistics tool.

### **2.3.1 Bug Synchronization Service**

NVIDIA's ChromeOS team has to work with two bug tracking databases: "Google Partner Bugs", Google's bug tracking database, and "NVBugs", NVIDIA's internal tracking database. A problem arises when changes such as the creation of new bugs or the addition of comments to existing bugs are made on the Google tracker. NVIDIA's engineers need Google's changes to be reflected in some manner on NVBugs as well, since the engineers use NVBugs to track the progress of each bug found in the company's products. NVIDIA needs a manner of synchronizing the changes made on Google Partner Bugs to NVBugs. These changes can include creation of new bugs, updates to existing bugs, or comments on existing bugs.

NVIDIA has synchronization services for other partners it works with, such as Microsoft, Apple, and Audi. The process of building a synchronization service starts with the team that works with the partner registering a request for a synchronization service. Once a request is made, the NVBugs

developer team works with the partner to find a “sponsor”; an employee of the partner who will devote some of their time to configuring the partner database to allow for data exchange with the NVBugs database. Once the databases are appropriately configured, the NVBugs team works in conjunction with the sponsor to develop any additional software layers necessary to establish two-way synchronization.

NVIDIA has tried several times in the past to establish a partner database synchronization with Google, but this has never worked because of an inability to find a sponsor from Google. NVIDIA’s Project Managers have had to devote a part of their workday to manually copying and pasting over any updates made on Google Partner Bugs to NVBugs in the absence of an automated system. This approach has several drawbacks:

- It is not an efficient use of the Project Managers’ time.
- It has the potential for delaying updates.
- It has the potential to be error-prone – an update could be missed during the copy-paste operation.

This copy-and-paste system provides a one-way synchronization from Google Partner Bugs to NVBugs. Though other partner synchronization projects provide two-way synchronization, the ChromeOS team only requires the capability to synchronize bugs one-way – from Google Partner Bugs to NVBugs – and thus the unidirectional nature of the current copy-and-paste system is not considered among one of its drawbacks.

NVIDIA is looking for a way to create a synchronization service without the sponsorship of Google. The major features of this service would be:

- One-way synchronization: all changes made on Google Partner Bugs that are assigned to NVIDIA are reflected in NVBugs.
- Automated: the service requires little to no interaction with an administrator.
- Periodic: the service runs on pre-configured time intervals.

### **2.3.2 Bug Statistics Tool**

Currently, NVIDIA’s PMs manually tabulate the data and generate the charts that provide them insight into a project’s status. This process involves running searches in NVBugs to find the bugs that they are interested in. They then examine each bug's status, and group the statuses by date to obtain a view of how the bug statuses are changing over time. They also note the engineer assigned to each bug to tabulate bugs per engineer. Once they have this data, they transcribe it into a format that a charting

application, such as Microsoft Excel, can use to generate charts. Finally, they configure the charting application to generate the charts that are required. This process is repeated each time fresh statistics are required. Manually generating these statistics is time-intensive and can lead to inconsistency in the charts generated for different projects.

NVIDIA is looking for a tool that would automate the generation of bug statistics, and would display the trends for different projects as determined by the user.

## 3 Background

### 3.1 Bug Trackers

NVIDIA has a wide portfolio, creating many products for various companies, such as Microsoft, Audi, Apple, Google, etc. (Section 2.1). Many teams are created to work on each of these products, and each of these teams needs to be able to track its progress from day to day, and across multiple timezones. Many methods exist for tracking progress, whether through deliverables, deadlines, or simply keeping track of what is accomplished and what needs to be done. A separate way of tracking progress is through bug tracking.

Bugs are defects that exist in code or in hardware that someone will need to fix. Many project managers use bugs to track progress, as a product that is close to being in a deliverable state will have few bugs actively remaining.

Teams use bug trackers to keep track of bugs. These bug tracking databases contain bugs that are submitted by developers, Quality Assurance teams, or anyone who is testing the product. The database also keeps track of relevant bug information, such as a priority level, a description of the bug, who submitted the bug, and who is working on fixing the bug. The database enables development teams to keep track of all bugs that have been fixed, are being worked on, and still need to be looked at. Team members are able to estimate a product's development status through perusing the database, and seeing how close to being near bug-free it is.

#### 3.1.1 NVIDIA Bug Tracking

With NVIDIA's large product portfolio, there are many development teams working on different product development lines. NVIDIA developed an internal bug tracking database, NVBugs, to aid in bug tracking within each individual product development line. NVBugs is used across all of the departments NVIDIA consists of (Software, Hardware, System Engineers, Information Systems, and Platform Software), with each department possessing their own division of the database. Each individual division is broken down into modules, with each team able to have their own module for their bugs. This allows users the simplicity of examining bugs only in a certain module. Bugs are able to be shared across multiple modules, allowing multiple teams to access the bug simultaneously. There is also a dedicated NVBugs team, whose job it is to update, manage, and test the NVBugs platform [61]. The NVBugs team

has created two SOAP APIs (Section 3.6.2) for programmatic interaction with NVBugs: an Internal API, and a Partner Integration API.

### **3.1.1.1 NVBugs SOAP APIs**

NVBugs allows direct access to its database through custom SOAP APIs [55] (Section 3.6.2). These APIs allow users to write scripts that can query the database and create, edit, examine, or comment on bugs. Bugs are unable to be deleted through the SOAP APIs. The APIs use the SOAP protocol to send and receive bug objects to and from the database. There are two APIs, the Internal API and the Partner Integration API, and they differ mainly in the functionality they expose.

#### **3.1.1.1.1 Internal API**

The Internal API is meant to be used by NVIDIA engineers for quick modification of their bugs via scripts. The four main functions that the Internal API supports are:

- *GetBug*: used to search for a specific bug using the bug's unique ID, and will produce a bug object.
- *GetBugs*: uses a raw XML query to search through the database for a set of bugs that match that query. *GetBugs* will return a list of bug objects that satisfy the query. Only the following fields are searchable using *GetBugs*: *ApplicationDivision*, *BugID*, *Synopsis*, *Description*, *Requester*, *Engineer*, *QAEngineer*, *AnyEngineer*, *ARB*, *DevTech*, *BugType*, *BugAction*, *Disposition*, *Module*, *TimeSpan*, and *GTLTaskID*.
- *SaveBug*: saves a bug object to the database. If the bug object's ID is zero, it will create a new bug and give it a unique ID. If the bug object's ID is non-zero, it will update the existing bug with matching ID, using the given bug object's information.
- *AddComment*: adds a comment from a user to a bug in the database.

#### **3.1.1.1.2 Partner Integration API**

The Partner Integration API is meant to be used by the NVIDIA teams that work with both the NVIDIA bug database and a partner company's bug database (Section 3.1.1.3). The three main functions that the Partner Integration API supports are:

- *GetPartnerBugs*: used to get NVIDIA partner bugs that have been modified since the last time they were synchronized with a partner.

- *SavePartnerBug*: used to save a Partner Bug object to the database. A Partner Bug object uses *PartnerInformation* tags that are specific to each partner (Section 4.2.3.1). The usage of these tags differentiates a Partner Bug from a normal bug object.
- *MarkBugsForExport*: used to force the synchronization of an NVIDIA bug with a partner's database, even if the NVIDIA bug has not undergone any updates since the last time it was synchronized.

### 3.1.1.2 NVBugs User Interface

NVBugs has a front-facing user-interface (UI) that allows anyone with the proper credentials access to the database of bugs. The UI allows for people to interact with the NVBugs database without having to write scripts that interact with the SOAP API. The UI contains a “View” system that allows for very complex searches to be made. These Views will display all bugs that match the View’s search query. Users are able to create a custom View for quick access to bugs that pertain to them. Figure 1 shows a custom View named “NVIDIA ChromeOS Bugs” that displays all bugs in the NVIDIA database that are of interest to the ChromeOS team.

BuildID	Synopsis	ARB	Bug Action	Date Created	Disposition	Engineer	Module	QA Engineer	My Notes
200084154	[T124/Chrome/Kitty/Google Hangout][GVC] Out of Memory observed randomly for 1.5 user participant video hangout call	Gr	Dev - Open - To Fix	03/02/2015	Open issue	Gr Shivas Patel	Mobile_Chrome_Multimedia	Gr Abhilekh Tiwari	
200084508	[T124/Chrome/Kitty/Google Hangout][GVC] Video quality keeps on flickering randomly	Gr	Dev - Open - To Fix	03/02/2015	Open issue	Gr Viranjan Pagar	Mobile_Chrome_Multimedia	Gr Abhilekh Tiwari	
200084582	[T124/Chrome/Kitty/Google Hangout][GVC] Couldn't start video call because of an error observed on few Kitty devices during Hangout video call	Gr	Dev - Open - To Fix	03/02/2015	Open issue	Gr Shivas Patel	Mobile_Chrome_Multimedia	Gr Abhilekh Tiwari	
200084552	[T124/Kitty/Google Hangout][GVC] Video Conference stalls for some devices during active video hangout call, whereas encode works	Gr Rahul Marathe	Dev - Open - To Fix	03/02/2015	Open issue	Gr Viranjan Pagar	Mobile_Chrome_Multimedia	Gr Abhilekh Tiwari	
200084936	[Dragon/A44] Investigate display clock rate for power saving	Gr	Dev - Open - To Fix	03/02/2015	Open issue	Gr Hari Mudaliar	Mobile_Chrome_Kernel	Gr Abhilekh Tiwari	
200083663	[T124/ChromeOS/AIO_Kitty] Kitty/AIO login to LG Chromebook[Intel Haswell] in Perf KPI Time to Resume and Boottime cases by ~44%, Build Used: CPFE_R42_6787.0.0.LDN-10.025	Gr	Dev - Open - To Fix	02/27/2015	Open issue	Gr Emily Jiang	Mobile_Chrome_Kernel	Gr Abhilekh Tiwari	
200083657	[T124/ChromeOS/AIO_Kitty] Kitty/AIO login to LG Chromebook[Intel Haswell] in Perf KPI IO-Operations by a margin of ~85%, Build Used: CPFE_R42_6812.0.0	Gr	Dev - Open - To Fix	02/27/2015	Open issue	Gr Emily Jiang	Mobile_Chrome_Kernel	Gr Abhilekh Tiwari	
200083633	[T132/ChromeOS/A44] Add VBI decode support in libtegra42 for SVIDA interface	Gr	Dev - Open - To Fix	02/27/2015	Open issue	Gr Viranjan Pagar	Mobile_Chrome_Multimedia	Gr Abhilekh Tiwari	
200083575	[T124/ChromeOS/AIO_Kitty] [Google Partner Bug 37250] Splash screen stays on a bit longer than expected resulting in higher boot time values, Build used: CPFE_R42_6812.0.0	Gr Titan Lee	Dev - Open - To Fix	02/26/2015	Open issue	Gr Isaac Simha	Mobile_Chrome_Google	Gr Abhilekh Tiwari	
200083324	[T132/A44/Dragon] dragon: switch to and enable JOMMU-backed DMA mapping API	Gr Mark Zhang	Dev - Open - To Fix	02/26/2015	Open issue	Gr Joseph Lo	Mobile_Chrome_Kernel	Gr Abhilekh Tiwari	
200083203	[T124/Nyan/Chrome][Google Partner Bug 37096] GPU process crash if tried to play back to back videos using SVIDA interface	Gr Titan Lee	Dev - Open - To Fix	02/25/2015	Open issue	Gr Shivas Patel	Mobile_Chrome_Google	Gr Abhilekh Tiwari	
200083854	[T134/Chrome/Intel] Performance for 720p decode+encode optimization	Gr	Dev - Open - To Fix	02/24/2015	Open issue	Gr Viranjan Pagar	Mobile_Chrome_Multimedia	Gr Abhilekh Tiwari	

Figure 1: “NVIDIA ChromeOS Bugs” Custom View on NVBugs  
Reference: [81]

The UI also includes a *Buggle* search bar, an internally developed search bar for NVBugs. This search bar is the UI equivalent of *GetBug* and *GetBugs*. If a user inputs a bug's ID, the UI will display the bug with that ID. If a user inputs a search query, the UI will display a list of the bugs that match that query, similar to the *GetBugs* function.

Users can view individual bugs in the UI. The user can edit and add comments in this display, if the user has permissions to edit the bugs. Users can also create bugs using an empty template.

### **3.1.1.3 NVBugs Third-Party Integration**

Many of NVIDIA's teams work with third-party customers. These teams develop graphics cards, kernels, and other software and hardware for these customers, and need to track bugs during this production. Many of the customers have their own bug tracking database. The customer company's QA teams can submit bugs to these internal bug tracking databases. The NVIDIA team has access to the third-party bug tracker database to communicate with the customer's QA team. However, the NVIDIA team also needs to port the bugs over to NVBugs, in order to discuss the actual fixes internally.

The NVBugs team used the SOAP API (Section 3.6.2) to create a third-party bug tracker integration tool [54]. The team finds a sponsor from the customer side (usually someone within their bug tracking database team), and works with the sponsor to create a synchronization tool that synchronizes any relevant bug from the customer's database with the NVBugs database. The sponsor implements a method on their own database to send SOAP messages to the NVBugs. A SOAP message containing the bug information is created each time a bug is added or updated on the customer database. The SOAP message is then sent to the NVBugs database, where the message is processed, and the relevant actions are executed. If the customer company does not produce a sponsor, the NVIDIA team will manually add and update the bugs, usually through copying and pasting directly from the customer's database UI.

### **3.1.2 Google Partner Bugs**

The ChromeOS team at NVIDIA works on the Tegra processor [59] that is used in Google's Chromebooks [28]. Thus, a number of the bugs that are submitted to or found by Google have to be worked on by NVIDIA's engineers. The bugs that are sent to the NVIDIA ChromeOS team from Google are referred to as Google Partner Bugs.

### **3.1.2.1 Google Project Hosting**

Google provides the Google Project Hosting service for interacting with Google Partner Bugs via a web interface [32]. The Google Project Hosting web interface for Google Partner Bugs is not publicly accessible. However, Google Project Hosting also hosts other open-source projects, most of which are accessible to the public. We used the Chromium project to gain familiarity with the Google Project Hosting platform [31].

#### **3.1.2.1.1 Chromium**

The bugs for the Chromium open-source project are hosted on Google Project Hosting [30]. The web interface allows users to browse through the list of all bugs, referred to as “issues” in Google Project Hosting. Users can examine the details of an individual issue, including current status, comments, and activity.

We were primarily interested in the different methods available to export these bugs from the Google Project Hosting interface, so that we could then update the NVIDIA bug database accordingly. Google Project Hosting does not offer a SOAP API, at least not one that is publicly available or documented. We found that the Chromium project offered three options for exporting bugs: an Atom feed, export to a comma-separated values (CSV) file, and email updates.

##### **3.1.2.1.1.1 Atom Feed**

Rich Site Summary (RSS) is a method of publishing news and updates in an automated fashion, by using Extensible Markup Language (XML) that is readable by computers. An RSS document, called a “feed”, contains text and metadata. Atom is a newer format of providing XML updates, with less restrictive licensing, support for XML namespaces, and some other minor differences [95][46].

The Chromium project offers an Atom feed that contains information about the latest changes or comments made to existing bugs, and also includes information about any new bugs that are filed. The feed contains complete information about these updates, and can thus be a viable manner of monitoring changes made to the Chromium Google Project. However, the feed displays just the last 20 updates made to the project’s bugs. Using the Atom feed would require that we track the time we query the feed and ensure that we do not lose any updates if there are multiple changes made within a short span of time.

#### 3.1.2.1.1.2 *Comma Separated Values File*

Comma-separated Values (CSV) is a common file format for storing tabular data in plain-text form. A CSV file can contain any number of rows of a table, with each row separated by line breaks. Within each row, the data pertaining to different columns is separated using commas [43].

The Google Project Hosting interface allows the user to export bugs to a CSV file. However, this option comes with two caveats.

First, the CSV file generated is truncated to just the first 100 results. The final line in the file contains a link to another file with the next 100 results. This means that obtaining more than the first 100 bugs would require multiple queries and would result in multiple files.

Second, within the CSV file itself, we observed the following column headers [29]:

- ID
- Pri
- M
- Week
- ReleaseBlock
- Cr
- Status
- Owner
- Summary
- AllLabels
- OS
- Modified
- ModifiedTimestamp

None of these columns included information about the comments made on the issue, which is a necessary piece of data to be synchronized. The CSV generator is a pre-made tool provided by Google Project Hosting, and does not provide any capability for modifying what data is exported.

#### 3.1.2.1.1.3 *Email Subscription*

The third option for exporting bugs is the email subscription service. Google Project Hosting generates an email when an issue is created, updated, or commented on. The email is sent to each email

address that is listed under the "CC" field for that particular bug. The CC field can also include mailing lists.

An email address – either a personal email or a mailing list address – can also be added to the list of subscribers for the overall project. Project subscribers are automatically added to the CC list for every bug under that project.

The generated email contains the following information:

- the email of the author of the change
- the content of the change, including both comments and other updates
- the Google issue ID

Email subscription offers an easy manner of getting updates for both new and existing bugs.

#### 3.1.2.1.2 Export Options for Google Partner Bugs

The Google Partner Bugs project does not offer the Atom feed option for exporting bugs that was available using the Chromium project. However, the Google Partner Bugs project does support both CSV and email subscriptions. The limitations of CSV files observed in the Chromium project (Section 3.1.2.1.1.2) – not containing comment updates, and not displaying all bugs in a single file – also apply to the Google Partner Bugs project. The email subscription option exposes the same information as seen in the Chromium project. Exporting to CSV and email subscriptions are both viable options for what we are trying to accomplish.

## 3.2 Email Protocols

We investigated the email subscription service offered by Google Project Hosting as a way to track updates to issues in a project. Three standard protocols were researched as potential ways to interact with the emails generated by the subscription service: Simple Mail Transfer Protocol (SMTP), Post Office Protocol – Version 3 (POP3), and Internet Message Access Protocol (IMAP) [45].

### 3.2.1 SMTP

SMTP is a protocol used for sending emails between servers [80]. Python has a module, *smtplib* [71], which provides basic functionality to send email using the SMTP protocol. However, if we were to implement a solution using the Google Project Hosting email subscription service, we would need to be able to read incoming emails. Since SMTP cannot be used to read email, this protocol alone will not provide a viable solution to our problem.

A second Python module, *smtpd* [70], defines functionality to implement a custom SMTP server. The *smtpd.SMTPServer* object can be used to listen on a port for incoming emails. Using this object, we could receive emails from the subscription service and perform the corresponding updates to NVBugs in real time.

### **3.2.2 POP3**

The POP3 protocol is used for email retrieval. A device using this protocol retrieves email waiting on a server and downloads it locally for offline management [64]. A downside to being able to manage email offline without constant connectivity to the internet is a loss in synchronicity with other devices. Once the mail has been downloaded, it is no longer kept in sync with the email server. If an email is deleted from one device's inbox folder, the email will still be in the inbox when accessed on a different device. This lack of synchronicity is not suitable for systems with multiple devices accessing the same email. The POP3 protocol does provide the necessary ability to read incoming emails, making it a viable potential solution to our problem.

### **3.2.3 IMAP**

IMAP is a protocol used for accessing and manipulating emails and mailboxes on a server [44]. The protocol is useful in situations when mail will be accessed and manipulated on multiple devices. Any change to the mailbox, such as deleting an email or renaming a folder, is reflected on the server and therefore on all devices accessing the mailbox. In addition, emails can be deleted without downloading the entire email first, which reduces usage of time and resources. However, this direct manipulation of the server requires a device to be constantly connected to the internet while working with the mailbox, which may not be feasible or practical. The IMAP protocol could be used as a viable solution for reading update emails from the Google Project Hosting subscription service.

## **3.3 Web Scraping**

The Google Project Hosting user interface provides another potential source for data on Google Partner Bugs. The interface consists of many web pages composed of Hypertext Markup Language (HTML). "Web Scraping" is the practice of using computer programs to parse HTML pages and gather information [79].

### 3.3.1 HTML

HTML pages are structured text documents consisting of a hierarchy of elements, each containing a name, attributes, and content [87]. Most elements contain the name and attributes in a “start” tag, followed by the content, followed by an “end” tag:

```
<name attributes>content</name>
```

Each tag is marked by using surrounding angle brackets. End tags contain the name of the element prepended by a backslash. One commonly used HTML element is the “paragraph” element:

```
<p align="right">A paragraph is written here.</p>
```

#### Code Fragment 1: The Paragraph HTML Element

The name of the element is “p”, an attribute that it has is “align=‘right’”, and the content is “A paragraph is written here.”

Some attributes can provide extra information as to what the content will be. The “title” attribute, for example, is used to specify extra information which is often shown in a tooltip when the mouse hovers over the element [93]. Other attributes, such as “class” and “id”, are not specifically meant to provide information on the content, but humans can often infer what the content will be based on their names.

### 3.3.2 Google Project Hosting HTML Structure

The following is an HTML fragment from Google Project Hosting that defines a comment made on a bug:

```
1    <div class="cursor_off vt issuecomment" id="hc1">
2        ...
3        <span class="date" title="Tue Nov 25 15:23:41 2014">
4            Nov 25, 2014
5        </span>
6        ...
7        <pre>
8            Hi Ken,
9            We found out that the calculations in the driver...
10       </pre>
11    </div>
```

#### Code Fragment 2: HTML Fragment from Google Project Hosting Reference: [35]

On line 1, one of the values of the “class” attribute is “issuecomment”, which implies that this tag starts a comment. Line 3 shows the date and time that the comment was made, again implied by the class attribute, and lines 7-10 contain the text of the comment that was entered by the user. These inferences about the HTML structure must be made by a human, but once the structure is determined it can be used to programmatically traverse and find specific data.

### **3.3.3 Retrieving and Parsing HTML**

The HTML for web pages can be retrieved using simple HTTP GET requests. Each bug in Google Project Hosting has its own page of information. This organization means that at least one HTTP request needs to be made for each bug, adding complexity to the system. Once the HTML has been retrieved, there are many Python modules available that can be used to parse the data into different data structures.

## **3.4 Automation**

Performing a task always has an associated cost. Companies perform a large number of tasks for different purposes according to their business strategy. Companies can increase the amount of tasks they complete if they minimize the cost associated with each task. One of the most pressing costs is time.

A common way of addressing the cost of time is through automation. Automation usually breaks down into two parts: actually performing the task, and repeatedly completing the task. In order to automate the actual performing of the task, the user will often write a program or script that will process ordered instructions and complete the task without input. The user then needs a way to repeatedly run this program: either the user manually starts the program each time, or has it automatically start on its own.

### **3.4.1 Daemon**

A daemon is a computer program that runs as a process in the background of the computer, usually out of the view of the user [73]. Daemons are used to automate processes, such as starting programs after booting the computer, or checking for updates. Daemons are usually self-sufficient, meaning they start manually or automatically, run their instructions, and then close without input. Daemons split off processes from the main system, which allows for multiple daemons to run at the

same time. Daemons will not restart on their own if they fail, however, nor will they be able to start without being prompted from a different process.

#### **3.4.1.1 Cron Jobs**

A “cron” is a time-based software utility similar to a daemon that is used to schedule jobs [92]. “Cronjobs” run on a schedule based on the computer system’s time. The *cronjob* will perform its task every time it is scheduled to by the programmer. *Cronjobs* allow the automation of other tasks, enabling the tasks to run on a schedule and removing the need for a user to manually start the task each time. If a task prompted by a *cronjob* fails, the task will start again the next time the *cronjob* is scheduled.

#### **3.4.1.2 Run-one**

“Run-one” is an Ubuntu package that ensures only one instance of a command with specific arguments is running at a time [82]. When automating a process to run on a certain interval, like with *Cron* (Section 3.4.1.1), it is important to take into account the potential runtime of the process. If the runtime of the process is always the same, we can safely assume the length of the interval on which to run the process. For example, if the process always takes exactly 10 seconds to run and we would like to run the process as often as possible, we could specify an interval as short as 11 seconds. However, if the runtime of a process varies, the interval at which the process can be run is not as clear. If the interval is too short, there is risk of starting a second instance of the process before the first is done. This overlapping of processes can cause unexpected behavior, especially if the process accesses or modifies external data.

The *run-one* command takes another command and set of arguments as its arguments. It checks if there is already a process running for the given command and arguments. If there is, *run-one* silently exits. If there is not, it allows the given command and arguments to be executed. *Run-one* can be used to prevent overlapping processes when automating a process with a variable runtime.

The *run-one* package is developed to be used with the Ubuntu operating system, however, we found that it was also compatible with the CentOS operating system. Ubuntu and CentOS are Linux distributions and therefore the systems are very similar (for information on the differences between CentOS and Ubuntu, see [19]). Software can be compatible with both, but will require different processes for packaging and installation [13]. We were able to install and use *run-one* on a CentOS machine with no changes to the program’s packaging or configuration.

## 3.5 Security

A new and pressing concern with the advent of computing is the concept of online security. In the past, security consisted of protecting physical assets. No thief could steal items as long as the thief never got access to restricted areas. No spy could steal ideas unless an employee purposefully exposed secrets. Security changed once the Internet was brought into use, as now someone could steal important files from halfway across the world with a simple click.

### 3.5.1 Encryption

One of the biggest weaknesses a computer system can have is that users, at some point, will need to access it. A computer system is intended for only valid users to access it, by the user providing valid credentials. This prevents unknown users from using the system, as the unknown user will lack valid credentials. However, if the unknown user has the valid user's credentials, then the unknown user will gain access to the system. The most secure way to deal with this weakness is to ensure that the unknown user can never get the valid user's credentials.

A valid user wanting to automate a task (Section 3.4) may run into a problem when the automated process requires valid credentials to be entered. The valid user can get around this by manually entering the credentials each time, but this removes the usefulness of automation. The valid user can simply provide the automated task the valid credentials needed, but this runs a large security risk. If an unknown user gains access to the automation process's files, the unknown user can view the valid user's credentials, giving the unknown user access to the system.

A solution to an unknown user gaining access to valid credentials is encryption. The valid user can encrypt the credentials, and the automated program will decrypt and use the credentials as it runs. An unknown user viewing the encrypted credentials would not be able to decrypt the credentials and would be unable to access the system.

#### 3.5.1.1 *Gnu Privacy Guard*

Gnu Privacy Guard (GPG) [24] is a free implementation of a two-key encryption service, Pretty Good Privacy (PGP) [10]. A key is a way of decoding an encrypted message. GPG works by having users generate their own private key, exchanging keys, and using the other's key to encrypt and decrypt a message. GPG uses a simple step-by-step process:

- User A generates a private key.

- User A encrypts his/her private key, and publishes the encrypted key on the Internet. The published key can only be decrypted with User A's private key.
- User B downloads User A's encrypted key.
- User B encrypts a message for User A using User A's encrypted key, as well as User B's own public key.
- User B sends the encrypted message to User A.
- User A first decrypts the message using User B's public key, and then using User A's own private key.
- User A now has the decrypted message User B sent.

## **3.6 API Protocols**

An Application Programming Interface (API) is a set of protocols and functions for interacting with software applications. An API should be platform-agnostic and language-agnostic to allow for maximum compatibility. To achieve this purpose, a number of industry standard protocols are used in the design of an API. An overview of each protocol we used is contained in this section.

### **3.6.1 XML**

A number of the protocols we used are based off Extensible Markup Language (XML), so we start by providing an overview of what XML is.

XML is a document markup language. XML describes a set of rules used to encode documents in a format that can be read by humans, and parsed by computers [16]. An XML document is entirely textual in nature, and is represented by a string of characters.

A sample XML snippet can be observed in Code Fragment 3.

```
<?xml version="1.0" encoding="UTF-8"?>

<note>

    <to> Tove</to>

    <from>Jani</from>

    <heading>Reminder</heading>

    <body>Don't forget me this weekend!</body>

</note>
```

**Code Fragment 3: Sample XML Snippet**  
Reference: [76]

XML is a manner of describing data, and supports different human languages via the use of Unicode (Section 3.6.4) [96]. Many document formats have been developed using XML formats, including RSS/Atom (Section 3.1.2.1.1.1) and SOAP (Section 3.6.2). XML's support for different languages and its prevalence as a format of machine-readable data interchange have led it to become a commonly used format for exchanging data over the Internet [89].

The tags used in an XML document are not predefined. There is also no constraint on what the tags can be. The author of an XML document can define his or her own arbitrary tags.

### **3.6.2 SOAP**

Simple Object Access Protocol (SOAP) is a protocol used to access application services over the Internet. Different programs need to be able to communicate over the Internet with one another in order to accomplish their tasks. There are a number of factors to be considered when designing a protocol for allowing such communication:

- The programs may be running on different operating systems.
- The programs may be designed using various different programming languages.
- The programs may be written for different technologies and have different runtime environments.

SOAP was created as a protocol that allowed communication between programs, taking all of the above factors into consideration. SOAP uses the Hypertext Transfer Protocol (HTTP) for communicating between applications [15].

A SOAP message is an XML document, with the following elements:

- Envelope: required; describes the document as a SOAP message
- Header: optional; informs the recipient about how to process the message
- Body: required; contains the actual SOAP message for the recipient
- Fault: optional; holds information about errors

A sample SOAP message skeleton is:

```
<?xml version="1.0"?>

  <soap:Envelope
    xmlns:soap="http://www.w3.org/2001/12/soap-envelope"
    soap:encodingStyle="http://www.w3.org/2001/12/soap-encoding">

    <soap:Header>

      ...

    </soap:Header>

    <soap:Body>

      ...

      <soap:Fault>

        ...

      </soap:Fault>

    </soap:Body>

  </soap:Envelope>
```

**Code Fragment 4: Sample SOAP Message Skeleton**  
Reference: [75]

An API that offers services over the Web can use SOAP to interact with the client.

### 3.6.3 WSDL

Web Service Definition Language (WSDL) is an XML language for describing Web services. A WSDL describes a Web service in terms of the messages it can send and receive. A WSDL is a contract – the service promises to accept and return messages that conform to its WSDL specification [18].

A WSDL can specify a range of interactions that are possible for the service. The client using the service can decide which of these interactions, if any, it wants to initiate. The WSDL describes how the interaction will occur once initiated by the client.

An example of a WSDL can be found in Appendix A. A Web API can expose its functionality by publishing the WSDL that it adheres to.

### 3.6.4 Unicode

Unicode is an industry standard for the encoding and representation of text in computing systems [42]. There are two major reasons as to why an industry standard for character encoding is important. First, there are many different languages used by humans across the world. Cumulatively, these languages encompass many thousands of characters and symbols. Computers need to be able to store and represent these character sets. Computers store characters by assigning unique numbers to each character. It would be ideal if a single encoding was able to provide character-to-number mappings for all characters that could possibly be used [91].

Second, when multiple encodings were used in different parts of the world, programmers ran into issues where different encodings use the same number for representing different characters. This reduced the portability of programs [20].

Unicode provides a unique character-to-number mapping, irrespective of the platform and language being used. It provides for easy portability and internationalization of programs.

Sample encodings in Unicode [90]:

A	U+0041
Z	U+005A
{	U+007B
ö	U+00F6

An API can specify what character encodings it supports [74]. Unicode provides an encoding that different systems can understand, irrespective of platform or language. API's that support Unicode are able to have widespread usage in different geographic locations.

## 3.7 Logging

We wanted to be able to log messages for successful synchronization operations and when errors were encountered. We used the *logging* module built into Python for this purpose [67].

### 3.7.1 The Python *logging* Module

The Python *logging* module provides an application programming interface (Section 3.6) that allows developers to implement an event logging system. Events are messages that describe the state of the program at a certain point during the program's execution. Events can have an importance level attached to them; this importance level is variable and can be decided upon by the developers.

Logging has three main advantages over printing messages:

- filtering: logging allows the script administrator to easily specify the level of verbosity desired in the output; simple printing does not allow this control.
- separation of importance: logging allows for printing messages of different levels to different files, making it simple to view only the errors or warnings; printing does not allow the separation of messages of differing importance.
- integration: since the *logging* module is a standard library, the messages from the bug synchronization service could be easily integrated with messages from any third-party Python modules that we would use [67].

### 3.7.2 Event Levels

The Python *logging* module can log events at different importance levels. There are six predefined logging levels. Each logging level corresponds to a numeric value. The numeric values determine the relative importance of the levels, and are used in deciding which events propagate to which loggers. The six predefined logging levels and their numeric equivalents are listed in Table 1 [67].

**Table 1: Predefined Logging Levels in Python's *logging* Module**

Level Name	Numeric Equivalent
NOTSET	0
DEBUG	10
INFO	20
WARNING	30
ERROR	40
CRITICAL	50

### 3.7.3 The YAML Data Format

“YAML Ain't Markup Language” (YAML) is a data-serialization format used to describe data. It uses an indented layout structure and is similar to plain text, making it useful for generating configuration files [21]. Configuration files are easily modifiable by administrators who may not have experience with the actual programming language. YAML is a format that allows for the easy understandability and modification of configuration files.

An example of a YAML file is:

```
1)    ---
2)    invoice      : 34843
3)    date       : 2001-01-23
4)    bill-to:
5)        given    : Jane
6)        family   : Doe
7)    address:
8)        road      : Foo bar road
9)        city       : Worcester
10)       state     : MA
11)       postal    : 01010
12)    total: 4443.52
```

**Code Fragment 5: Sample YAML File**  
Reference: Adapted from [97]

The “---” at the top of the file is used to indicate the start of a file stream. The relationships between data are preserved using indentation: “invoice”, “date”, “bill-to”, and “total” are top level elements, and “bill-to” contains another structure as its value. As observed on line 8, string values do not need to be enclosed in quotes.

## **3.8 Statistics**

A product’s development cycle is measured in various ways, all of which determine how much progress has been made. Project Managers (PMs) use progress reports to view the progress made in a project, usually through visual displays, such as charts and graphs. PMs are able to share this information with other PMs, employees on their teams, and upper management, to help convey the status of a project.

### **3.8.1 ChromeOS Statistics**

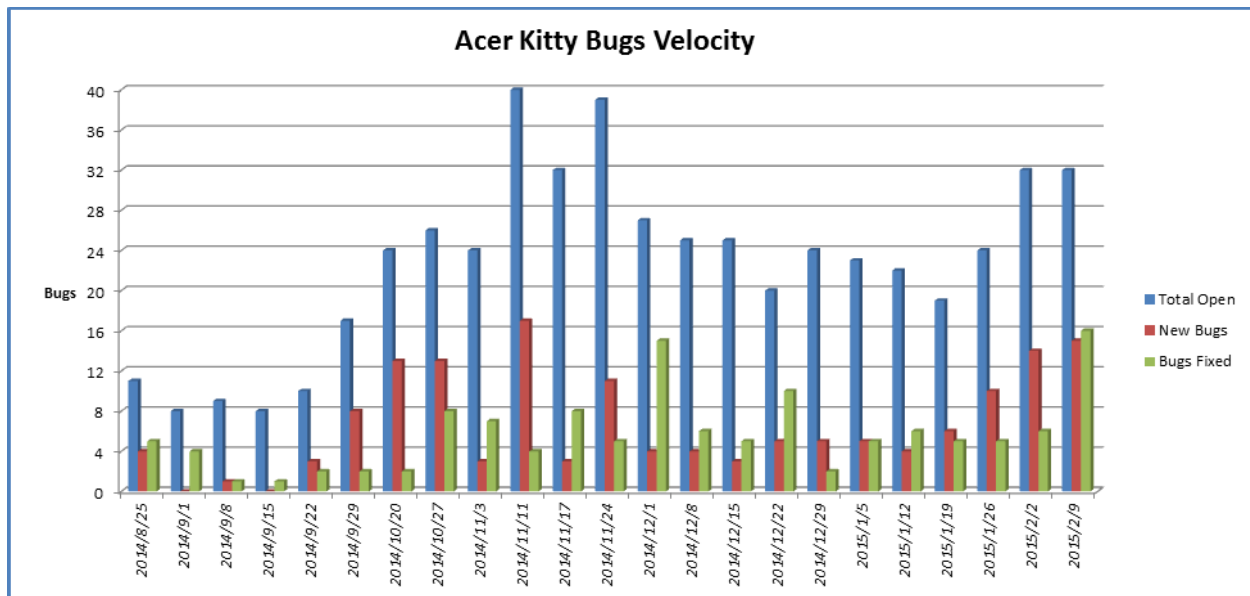
The Project Managers (PMs) of the ChromeOS team use bug trends to track the progress of the team. The fewer open bugs the team has, the closer the product is to being ready. PMs compare the rate at which new bugs are found against the rate at which the bugs are fixed to determine the stability of the product.

Before our project, the PMs viewed bugs in the NVBugs database using the web server API. The PMs would manually look through the modules and projects to find information on relevant bugs, and would then translate the information into statistics that could be shared. The three most common charts the PMs used are:

- Project Bug Velocity
- Project Bug Trend by Modules
- Project Bug Trend by Engineers

The charts can be observed in the figures on the following pages. Figure 2, Figure 3, and Figure 4 are charts that were used in a presentation by a PM of the ChromeOS project. Figure 2 and Figure 3 use the ChromeOS project ‘Acer Kitty’ as the target project, and Figure 4 uses the ChromeOS project ‘HP/Bailey’.

Figure 2 shows the Project Bug Velocity chart. The chart's columns display the number of open bugs, the number of new bugs, and the number of fixed bugs for a single project over multiple weeks. The chart shows the open/new/fixed levels for up to fifteen weeks in the past. This chart is used to show the velocity of the bugs for a single project: how many bugs are being created against how many are being fixed. This information allows PMs to make a rough prediction about when the number of bugs in the project will approach zero.



**Figure 2: Project Bug Velocity Chart**

Figure 3 shows the Project Bug Trend by Modules chart. The chart displays the number of open bugs per module for a single project over time. The numbers of open bugs for each module are stacked on one another, summing to the total number of open bugs. Each different module relating to the project is assigned a color, which is used to show the different amounts of open bugs. The graph shows the stacked modules for up to fifteen weeks in the past, allowing the viewer to see the differing number of open bugs for each module. This chart is used to compare the number of open bugs per module against other modules. This allows PMs to gain insight into how the different modules are tracking against each other with respect to reaching a state where the code can be released.

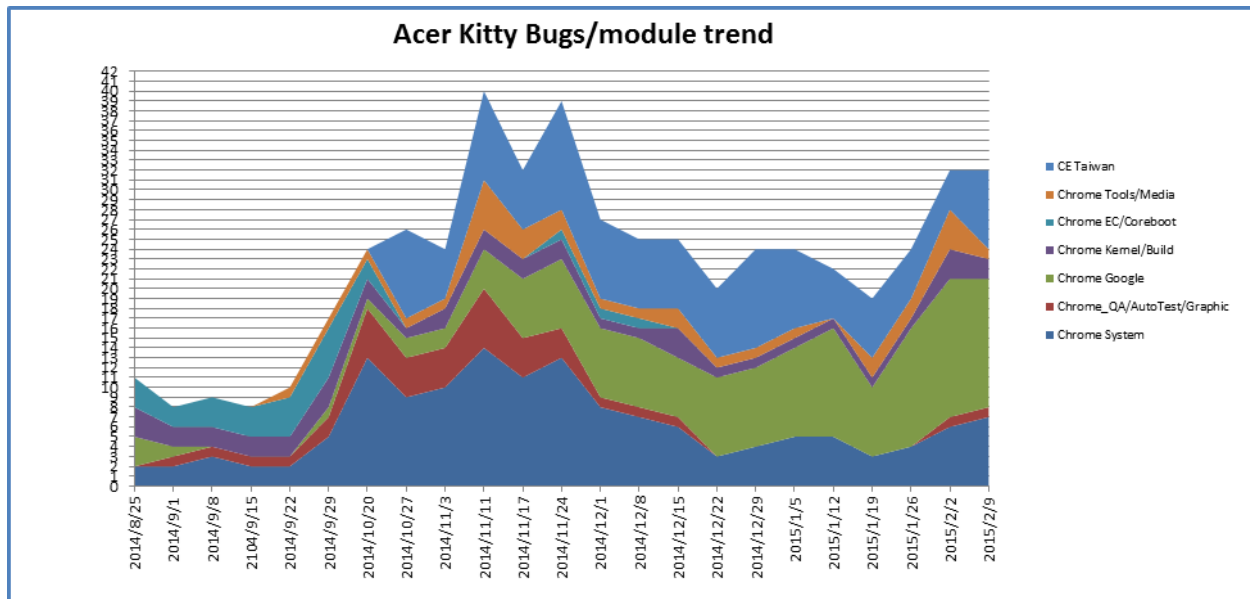


Figure 3: Project Bug Trend by Modules

Figure 4 shows the Project Bug Trend by Engineers. The chart shows the number of open bugs per engineer for a single project. The columns represent the number of bugs that are currently open and assigned to a specific engineer. This chart allows the viewer to see which engineers have the most number of bugs to fix for a particular project. This allows the PMs to make decisions about reassigning bugs to different engineers to ensure that workloads per engineer are roughly balanced.

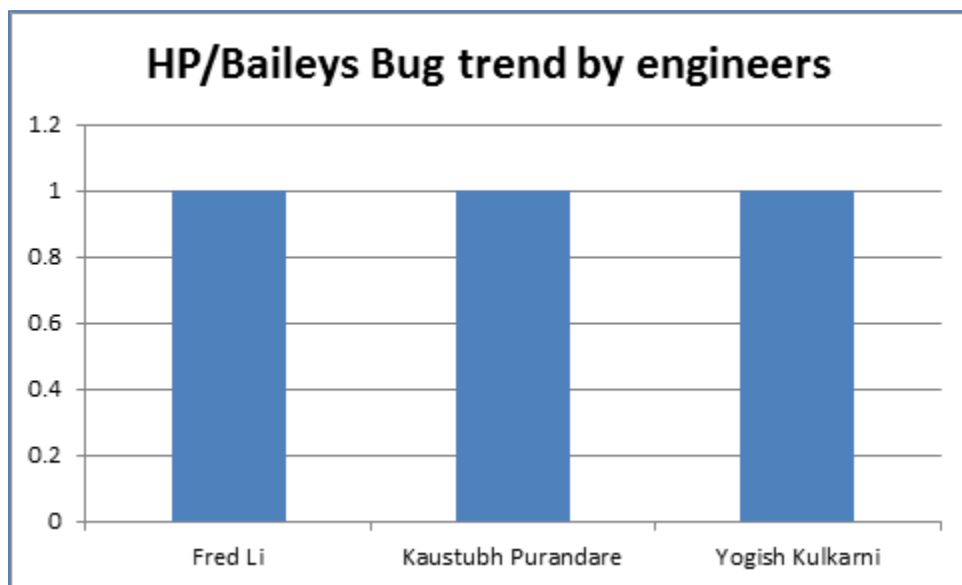


Figure 4: Project Bug Trend by Engineer

The PMs need to generate the above three charts for every project they are working with. Generating each chart takes a long time, as searching the bugs and recording the data is time intensive. A PM must search every module that they are interested in, and manually record each bug's info, such as request date, fixed date, days open, etc. The PM then uses this manually recorded data and generates a chart with the data using Microsoft Excel. This approach takes an enormous amount of time.

The web client of NVBugs does have several embedded charts. The embedded charts are limited in what they can display. The web client can also only display a single chart at a time. An example can be found in Appendix B.

### **3.8.2 Generating Dynamic Charts with Microsoft Excel**

The PMs currently generate the required charts by manually entering data into tables within Excel. This process is quite time-intensive and not the best use of managerial time.

Converting this process to be automatic would drastically reduce the time involved in generating the charts. This would have multiple benefits:

- PMs could better allocate their time.
- Charts could be generated at the click of a button, allowing PMs to access the latest statistics quickly and frequently.

Excel provides different methods of automating chart creation, one of which is Visual Basic for Applications.

#### **3.8.2.1 Visual Basic for Applications**

Microsoft Excel allows code to be run from a programming language called Visual Basic for Applications (VBA). VBA enables many Excel processes to be automated [50]. A user can use VBA to create a small series of instructions that Excel runs, called a macroinstruction (macro). These macros are able to be integrated into the Excel spreadsheets, and can be triggered through a user's input. Macros are able to automate table generation, mathematical formulas, and even document saving.

#### **3.8.2.2 Gremlins**

The Gremlins database is a backup database for NVBugs. The name "Gremlins" comes from an NVIDIA joke: "It is not in the best interests of a company to hire developers to create bugs. Because everything we do at NVIDIA is for the best interests of the company, the only logical explanation for why there are bugs is gremlins." [63].

Gremlins is a database that takes a snapshot of the NVBugs database every 15 minutes, and backs the data up. The difference between Gremlins and NVBugs is the API associated with these two services. NVBugs provides SOAP APIs that can modify bugs, as well as providing a web-client front end. Gremlins is used solely for information gathering, allowing SQL connections and queries. Gremlins allows users to programmatically pull specific information from NVBugs that the user would otherwise have to manually pull from the NVBugs web client.

## 4 Methodology

### 4.1 Integration with Google

We explored two different implementations for retrieving data from Google Partner Bugs: Email Scraping, and Web Scraping. After a partial implementation of both options, we chose to fully implement the Web Scraping strategy.

#### 4.1.1 Email Scraping

We chose to utilize the email subscription service offered by Google Project Hosting (Section 3.1.2.1.1.3) in our initial strategy for retrieving data from Google Partner Bugs. Google Project Hosting emails all project subscribers when a new bug is created or an existing bug is updated. Our bug synchronization service would act as a subscriber to the ChromeOS project, receive emails from Google Project Hosting, and parse those emails for useful information.

##### 4.1.1.1 *Setting up the Mailbox*

An email account, “bug-sync@nvidia.com”, was created and added to the list of subscribers for the ChromeOS partner project on Google Project Hosting. We set up the email account to automatically filter all emails received from “chrome-os-partner@googlecode.com” to an exclusive folder named “Google Bugs”. It was useful to separate the subscription emails from the inbox with an initial filter, since the “bug-sync@nvidia.com” email address could receive emails from outside the subscription service.

##### 4.1.1.2 *Accessing the Mailbox*

To access the “bug-sync@nvidia.com” mailbox, we chose to use the Internet Message Access Protocol (IMAP) discussed in Section 3.2.3. This protocol best suited our needs due to its persistence of mailbox changes to the server. If an administrator logged in to the mailbox using a web client such as Microsoft Exchange, it was important that the mailbox seen by the administrator reflect the state seen by the bug synchronization service. In addition, we used the state of the mailbox to determine the behavior of the bug synchronization service, so it was necessary to use a protocol that would allow us to make changes to the mailbox that would persist to the next time that the bug synchronization service ran. Post Office Protocol – Version 3 (POP3), discussed in Section 3.2.2, does not persist mailbox changes

to the server. In addition POP3 has no concept of folders or complex queries, meaning that we would be unable to retrieve only emails that had been filtered to the 'Google Bugs' folder.

Emails received from the subscription service were marked with an 'UNSEEN' flag in the 'Google Bugs' folder. When the bug synchronization service ran, it connected to the Exchange mailbox using the "imaplib" module offered in Python [66]. The service retrieved all emails with 'UNSEEN' flags from the 'Google Bug' folder, changing the 'UNSEEN' flags to 'SEEN'. These emails were then parsed to extract the necessary data to perform an update on the corresponding NVIDIA bug. If the update executed successfully, the email would be deleted from the mailbox. However, if the update was unsuccessful, the email would remain in the mailbox with the 'SEEN' flag.

Emails in the folder with the 'SEEN' flag would be ignored by the bug synchronization service on future runs. However, because the synchronization service does not delete those emails, an administrator could check the mailbox later and retrieve the data from the emails manually. A proposal for a future feature of this service was to be able to run the bug synchronization service in a mode where the service tries to update NVBugs using emails with the 'SEEN' flag as well as those with the 'UNSEEN' flag. This feature could be useful in a situation where the email format sent by the Google Project Hosting subscription service changes. A change in email format could cause the bug synchronization service to be unable to extract the necessary data from the emails, and cause updates from that synchronization attempt to fail. An administrator could be notified of this problem, adjust the program to account for the new email format, and run the synchronization service in the mode that includes emails with the 'SEEN' flag to retry those updates. A change in email format was expected to be very rare, making this feature a low priority.

#### ***4.1.1.3 Parsing the Emails***

Emails from the subscription service come in two formats, New Issue, and Update. An email in the New Issue format is sent by the Google Project Hosting subscription service when a user creates a new bug in the Google Partner Bugs project. The New Issue email format provided the following fields:

- Status
- Owner
- CC
- Labels
- Issue Number

- Author
- Synopsis
- Description
- Link to bug in Google Project Hosting

An email in the Update format is sent by the Google Project Hosting subscription service when an existing bug in Google Partner Bugs has a new comment, updates to the fields, or both. The Update email format provided the following fields:

- Updates, if present
- Issue Number
- Comment, if present
- Comment Number, if a comment is present
- Author
- Synopsis
- Link to bug in Google Project Hosting

Many of the data fields are unstructured text fields that are entered by a user when updating or creating a bug on Google Project Hosting. This lack of structure made it very important to choose a parsing strategy which relied as little as possible on formatting and would not be affected by special characters.

Both email formats shared a structure for the fields Issue Number, Author, Synopsis, and Link. This structure was prepended by 'New issue' for the New Issue format and 'Comment # [Comment Number] on issue' for the Update format. We used this structure as an identifying string to determine the email format:

[Issue Number] by [Author]: [Synopsis]

[Link to bug in Google Project Hosting]

We found the identifying string using a Regular Expression with the "re" module in Python, but avoided using Regular Expressions anywhere else due to their fragility [69]. Regular Expressions are used to search text for a pattern that you provide and return the location of that pattern. Some comments could contain information, such as commit logs, that included special characters that could cause a Regular Expression search to fail.

The identifying string was first followed by the Comment for an Update email, or the Description for a New Issue email, and finally the email footer which was the same for all email formats. We extracted the Comment/Description fields by taking a substring of the email from the end of the Link to the beginning of the email footer. The remaining fields on New Issue and Update were formatted in both emails as key-value pairs preceding the identifying string. Each key-value pair was formatted on its own line as 'Key: Value'. We were able to parse these pairs by separating each line as a pair and using the colon as the delimiter between keys and values.

#### **4.1.2 Web Scraping**

The Google Project Hosting Web interface provided another possible source for Google Partner Bugs information. Bug information on the website is displayed in two ways: a list of all bugs in the project, and a page of detailed information for each individual bug. Our bug synchronization service used web scraping techniques to utilize both data formats. More information about web scraping techniques can be found in Section 3.3.

##### **4.1.2.1 Google Authentication**

The *chrome-os-partner* project on Google Project Hosting is a private project with a specific list of users who have permission to view it. Our bug synchronization service provided user credentials in order to retrieve information from the site. The bug synchronization service email address, "bug-sync@nvidia.com", was added to the list of permitted users for the *chrome-os-partner* project. When the service ran, it used the Requests module [78] to start a session and get the Google Login page. The service then filled in the Email and Password fields and submitted the login form. The "Session" object in the Requests module allows certain parameters to be persisted across requests [77]. We used the Session object to create an authenticated session that persisted our login credentials. All requests made to Google during bug synchronization used the authenticated session.

##### **4.1.2.2 Retrieving Comma Separated Value Files**

The Google Project Hosting interface provided a view of a list of all bugs in the project, which could be filtered based on search criteria. The interface also provided a request URL to retrieve Comma Separated Value (CSV) files of all bugs in the list. The request URL contained two major parts: the query string, and the specified columns string. The query string specifies search criteria entered by the user. Only bugs that meet the search criteria are included in the CSV. The specified columns string specifies

which fields to include in the CSV. All CSVs included the following values by default: *ID, Pri, M, ReleaseBlock, Cr, Status, Owner, Summary, AllLabels*. We could customize this request URL and use it to request the CSV files directly from our bug synchronization service. Our service queried for all open bugs in the *chrome-os-partner* project. In addition to the default columns, we included the 'TimeModified' field, which provided the last time the bug had been modified. We also included the 'Proj' field, which provided the project the bug corresponded with.

Google Partner Bugs truncated each CSV file to 100 bugs. The 'start' parameter could be provided to the request URL to specify at which row to start in the CSV. If the parameter was omitted, the default value was 1. The first CSV would contain bugs numbered 1 to 100. To retrieve the next set of 100 bugs, we set the 'start' value equal to 101, and so on. We continued this process until we retrieved all bugs matching the given search criteria.

This process was used to retrieve a CSV file of all recently closed bugs in the *chrome-os-partner* project as well. We looked specifically for any bugs in the project that had been closed within the last two days. By including recently closed bugs, we ensured that any final changes up until being closed were captured and shown on the corresponding NVIDIA bug.

#### **4.1.2.3 Synchronizing the Bugs**

Once we retrieved the list of all open bugs in addition to the recently closed bugs in the *chrome-os-partner project*, we needed to determine which bugs had been modified since the last time they had been synchronized. A simple option would have been to run the service on a specified interval and use that interval to determine our assumed last modified time, for example once every hour. If the bug synchronization service runs once per hour, it would look for bugs that were modified in the past hour. However, this approach would not suffice for our system. Each bug that has new updates gets saved individually. Therefore, some bugs could successfully be saved, while others may fail, all within the same synchronization period. The service needed to know when each individual bug had last been successfully synced. The *GetSyncAudit* method in the Partner Integration API was made available for this purpose (Section 4.2.3.3). It provided an audit of all changes to a bug made specifically by our bug synchronization service.

Using the *GetSyncAudit* method, we could get the last time our synchronization service successfully updated an NVIDIA bug. If the bug had never been synchronized, we would create a new NVIDIA bug. If the bug had previously been synchronized, we compared the last time it had been

synchronized to the last time the corresponding Google Partner Bug had been modified. If the Google Partner Bug had been modified more recently, we then fetched all available information on the bug via web scraping (Section 3.3). We used this information to update the corresponding NVIDIA bug. A complete list of information retrieved can be found in Appendix C.

Closed bugs could have one of several statuses: *Assigned*, *Duplicate*, *Moved*, *Verified*, or *Won't Fix*. Bugs with the status *Moved* would no longer be in the *chrome-os-partner* database which meant that we would no longer have access to their corresponding web pages. We updated *Moved* bugs with only the basic information found in the CSV.

### **4.1.3 Comparing Strategies**

Both the Email Scraping strategy and Web Scraping strategy proved to work on a small scale. Associated costs and accuracy of synchronization needed to be considered for each system to make a decision on which to fully implement.

#### **4.1.3.1 Fragility of Email Scraping System**

We considered the different use cases of the email scraping system and found that there were many in which the email scraping system would fail. Using the email subscription service required that the email that handles the subscribers, “chromeos-partner-nvidia@chromium.org”, be in the CC field of a Google Partner Bug. The CC field of the bug can change overtime, and “chromeos-partner-nvidia@chromium.org” is often added to the CC list after the bug has been created, already with a set of comments and updates. The “bug-sync@nvidia.com” account would not receive emails for any of these comments or updates made before being added to the CC list and therefore would be unable to add those to the corresponding NVIDIA bug. In a similar case, “chromeos-partner-nvidia@chromium.org” could be on the CC list, subsequently removed from the CC list, and then added again. Similar to before, “bug-sync@nvidia.com” would miss a set of emails and be out of sync.

#### **4.1.3.2 Complexity of Web Scraping System**

The web scraping system provided all information available about a bug. However, retrieving this information involved many web requests. Each time the bug synchronization service ran, one request was made for each set of 100 bugs in order to get the complete CSV file. Another request was then made for the web page to scrape for each bug that needed to be updated. The number of web requests that needed to be made each time can be represented as:  $(X/100) + Y$ , where  $X$  is the number

of bugs in the CSV files and  $Y$  is the number of those bugs that have been updated. For example, if we retrieve 200 bugs and 100 of them have been updated, 102 requests would need to be made just to retrieve information on the bugs. A high number of requests will slow a process down, preventing the service from synchronizing often.

#### **4.1.3.3 Conclusion**

After observing the fragility of the partially implemented email scraping system, we compared the pros and cons to those of the web scraping system. While the web scraping process would be slower, it provided more accurate data. In addition, depending on the interval that the bug synchronization ran at, we expected the average number of bugs needing to be updated to be small, making speed less of a problem. The email scraping process would be less complex, but the system would be put out of sync if the service missed an email. The “chromeos-partner-nvidia@chromium.org” email, which is required to be in the CC field of a bug for our email service to receive updates, was often removed from a bug and added back to a bug later. This behavior made the risk level for missing emails and being put out of sync very high. We determined that the accuracy of the synchronization was the most important aspect, and decided to fully implement the web scraping system.

## **4.2 Integration with NVIDIA**

Once we parsed the bugs from Google Partner Bugs, we needed to interact with NVBugs. NVIDIA has two APIs built for NVBugs that we explored: the Internal SOAP API, and the Partner Integration API. The team was able to use both APIs to connect and interact with the NVBugs database.

### **4.2.1 Connecting to NVBugs**

We first had to connect to the NVBugs client in order to process any requests. The team connected to the web client of the API using the credentials of our bug-sync account. Connecting to the web client enabled the team to use the API’s function calls, which allowed us to communicate with the NVBugs database. We were able to connect to the NVBugs using each API. Different user accounts were needed for each of the APIs, however. The API used can be easily switched by changing the URL the client connects to.

#### 4.2.2 Using the Internal SOAP API

The first API we attempted to use was the Internal SOAP API. After connecting to the NVBug client, the service was able to call the Internal SOAP API's function calls. The service iterated through the bugs received from the Google Partner Bugs, taking a different action based on what kind of bug was being processed.

##### 4.2.2.1 *Saving a Bug*

The service called the API's *SaveBug* function if the bug being processed was a new bug. The API required XML wrapped in an enclosing pair of *WSBug* tags – we will refer to this as a “WSBug XML object”. Figure 5 displays an example WSBug XML object with all the fields that would be found on the NVBugs side. The WSBug XML object has a synopsis pulled from the Google Partner Bugs side, as well as a description containing the author of the Google Bug, a link to the Google Bug, and the description of the Google Bug. The rest of the fields in the NVIDIA database (such as priority and module) would be auto-generated with default values, or left blank. It is important to note that the Google Bug includes other fields that do not have a corresponding field in the NVBugs database. We could either not sync the information contained in those fields, or could add that information as a comment on the NVIDIA bug.

```

<tem:SourceBug>
  <tem:Synopsis>TEST NVBUGS SOAP - DEV</tem:Synopsis>
  <tem:BugID>0</tem:BugID>
  <tem:Description>Testing NVBugs soap - description here.</tem:Description>

  <tem:ManDays>10.000</tem:ManDays>

  <tem:FixNeededInProductionDate>2015-01-20</tem:FixNeededInProductionDate>
  <tem:Module>CUDA</tem:Module>
  <tem:ApplicationDivision>Software</tem:ApplicationDivision>

  <tem:RequesterFullName>Bill Phipps</tem:RequesterFullName>
  <tem:BugType>Software</tem:BugType>
  <tem:Version>TOT (NO BRANCH)</tem:Version>
  <tem:VersionFixedAfter/>
  <tem:FixNeededInBranch/>

  <tem:OperatingSystemList>N/A, Linux</tem:OperatingSystemList>
  <tem:GTLTaskIDs/>
  <tem:GTLTaskIDList/>
  <tem:PerforceCheckinNumber/>
  <tem:ARBFullName>Sam Chehab, Brian Ty, Michael Jansen</tem:ARBFullName>
  <tem:Disposition>Open issue</tem:Disposition>
  <tem:BugAction>Dev - Open - To fix</tem:BugAction>
  <tem:QAEngineerFullName/>
  <tem:EngineerFullName>Jitender Bisht</tem:EngineerFullName>
  <tem:DevTechFullName/>
  <tem:Priority>P0-Must have</tem:Priority>
  <tem:Severity>7-Task Tracking</tem:Severity>
  <tem:GeographicOrigin>CA-Santa Clara</tem:GeographicOrigin>
  <tem:Origin>Engineering</tem:Origin>
  <tem:Application/>
  <tem:ProjectedFixDate>2015-01-14</tem:ProjectedFixDate>
  <tem:ClosedDate></tem:ClosedDate>
  <tem:FixedDate></tem:FixedDate>

  <tem:CustomerName/>
  <tem:CustomerAdditionalList/>
  <tem:CCFullName/>
  <tem:IsSendNotification>false</tem:IsSendNotification>
  <tem:CustomKeywords/>
  <tem:PersonDaysSaved>0</tem:PersonDaysSaved>
  <tem:BangBuckFinalRatio>0.000000000</tem:BangBuckFinalRatio>
  <tem:Categories/>
  <tem:BusinessUnits>IT</tem:BusinessUnits>

</tem:SourceBug>

```

Figure 5: Example WSBug XML Object  
Reference: [55]

The WSBug XML object would have a *BugID* of '0'. The API interprets this as a new bug, and auto-assigns the new bug a unique *BugID* in NVBugs.

After the bug object was created, the WSBug XML object would be parsed into raw XML. The *SaveBug* function requires raw XML as a parameter. The *SaveBug* function then took the raw XML, and wrapped it into a SOAP message, giving it appropriate headers, envelopes, and format. This SOAP

message is then sent to the NVBugs client, which takes the message, parses it, and inserts the new bug into the database.

#### **4.2.2.2 Updating a Bug**

If the bug was marked for an update, the service would need to call the API's *GetBug* and *SaveBug* functions. If the *SaveBug* function is passed a WSBug XML object with a non-zero *BugID*, the API replaces the bug with the given *BugID* with the new WSBug XML object passed to the API. Another requirement for *SaveBug* to successfully update an existing bug is for the bug submitted to have all the same fields complete as the existing bug. The API provides a shortcut to verify the bug is complete, by pulling the existing bug from the database, and editing the changed fields, preventing the need to create a new bug from scratch. The *GetBug* function call takes in a *BugID* and returns that bug's WSBug XML object (Appendix D). The *GetBug* function call enables easy use of the *SaveBug* function call for updating existing bugs.

#### **4.2.2.3 Adding a Comment**

If the bug had comments that needed to be added, the service would call the API's *AddComment* function. The service takes the parsed comments, convert them into XML, and pass them to the function. This function takes in an existing *BugID* and the given XML, and adds the comment to the existing *BugID*. The author of the comment would always be the default system user, 'bug-sync', as that is the user submitting the comment. The user who submitted the actual comment on the Google side would have his/her name passed into the body of the comment itself.

#### **4.2.2.4 Problem with Internal SOAP API**

The biggest problem that we ran into with using the Internal SOAP API was the lack of a direct mapping between Google Partner Bugs and NVBugs, explained in Section 4.2.4.1. The lack of direct mapping forced the team to use a roundabout method to retrieve the related bug's *BugID* from the NVBugs database. The API uses the *GetBugs* (different from *GetBug*) function call to retrieve a list of bugs that match a search criterion. Our service creates an XML search query that searches for a unique identifier, and takes the *BugID* from the returned bug to be used in both updating a bug and adding comments. A problem arises from the fact that the object returned from the *GetBugs* function call is not an actual bug object, but simply an XML object. We could only receive the *BugID* from this object, and would be unable to use the object as the bug for the *SaveBug* call necessary in updating a bug. The

service would then be forced to make a second call to NVBugs using *GetBug*, to retrieve an actual bug object. We considered this a messy and expensive approach to the problem, and looked for other, cleaner ways to interact with the NVBugs database.

### **4.2.3 Using the Partner Bugs API**

We discovered the Partner Bugs API through discussions with Karthik Gopalakrishnan and Brian Ty. Karthik and Brian are NVIDIA engineers who work with NVBugs on a daily basis, and Brian wrote much of the original code for the NVBugs application. .

The Partner Bugs API is used by NVIDIA development teams who work with outside companies' bug tracking databases. The API enables a synchronization between the outside company's bug tracking database and NVBugs. The API is formatted in a way that exposes a bug's fields that are relevant for customers (such as *CustomerBugID*, *CustomerOwner*, etc.).

#### **4.2.3.1 Unique Partner Information**

Each partner using the Partner Bugs API has a unique *PartnerInfo*. *PartnerInfo* is a set of two fields, *PartnerGuid* and *PartnerName*, which are unique identifiers for each partner. Only bugs intended for that specific partner will have matching values. Partners without the proper *Guid* and *Name* will be unable to access other bugs.

#### **4.2.3.2 Interaction with NVBugs**

The Partner Bugs API uses the *SavePartnerBug* function call to both create and save bugs on the NVBugs side. When the service receives a bug object from the middleware, the service parses the bug into a *PartnerBug* XML object. The *PartnerBug* object is similar to a *WSBug* XML object, except the *PartnerBug* XML object contains customer fields, such as *CustomerBugID*, *CustomerOwner*, etc. The service combines the *PartnerBug* object with Google's Partner Info, and then the *SavePartnerBug* function call wraps the combined XML into a SOAP message. The SOAP message is sent to the NVBugs client, where it is parsed into the database.

When the NVBugs client reads the SOAP message, the client checks the *CustomerBugID* given against the list of *CustomerBugID*'s present in the NVBugs database. If the *CustomerBugID* is not listed, the client creates a new bug in the database, assigning it a new, unique *BugID*. If the NVBugs client finds an existing *CustomerBugID*, then the client updates the existing bug in the database, replacing all changed fields with the new fields passed through the API.

In the Partner Bugs API, comments are treated similar to a field. When creating the PartnerBug object, comments are added into an XML object inside of the PartnerBug object. These comments get passed through the API to the database, similar to other fields in the PartnerBug object. The API adds the comments to the existing comments for the bug in the NVBug database.

#### **4.2.3.3 *GetSyncAudit Method***

Another method exposed via the Partner Bugs API was the *GetSyncAudit* method. This method was added by Brian Ty (the lead NVBugs Developer) specifically for our bug synchronization service.

One of the issues we had to solve to use the Partner Bugs API was understanding when the NVIDIA bug had been last synchronized with the corresponding Google Partner Bug (the coding approach to this problem's solution is explained in Appendix E). This would allow us to understand if there were updates on the Google side that needed synchronization.

The *GetSyncAudit* method solved this problem. The *GetSyncAudit* method exposed the audit trail of all actions conducted by the synchronization service on the NVIDIA Partner Bug. The audit trail contained notes of each time a synchronization attempt was made. These notes logged the outcome of the synchronization attempt along with the Partner Bug's ID, as seen in Code Fragment 6.

```
Status: SUCCESS. Bug ID: 12345. Partner ticket #: 9876
```

#### **Code Fragment 6: NVIDIA Partner Bug Audit Trail Notes**

We were able to query the audit trail for each individual NVIDIA bug to obtain the last time it was successfully synced. We no longer needed to know the NVIDIA bug ID to use this query. We queried for all synchronization operations for the Google Partner Integration, narrowed down the search by specifying that we were looking for successful operations, and then narrowed the search further by specifying the Google Partner Bug's ID. This would return a list of all the successful synchronizations that had been conducted on the NVIDIA bug by our synchronization service. This list was ordered in descending order by time, so the most recent successful synchronization attempt was at index 0.

The *GetSyncAudit* method depends on the constraint that each NVIDIA bug will have either a unique Partner Bug ID, or no Partner Bug ID. If two NVIDIA bugs contain the same Partner Bug ID, the

method exhibited unpredictable behavior in terms of which bug it picked to return information about. We explore this scenario in further detail in Section 4.2.5.2.

#### **4.2.4 API Choice**

The usage of the various function calls available via both the Internal SOAP API and the Partner Integration API are explained in Section 3.1.1.1. We also examined these two API's from the perspective of solving a major challenge that we had encountered during the project – that of uniquely mapping Google Bugs to NVIDIA Bugs.

##### ***4.2.4.1 Mapping Google Bugs to NVIDIA Bugs***

A major problem that we resolved was the issue of mapping Google Bugs to the corresponding NVIDIA Bugs. Upon receiving updates from the Google Partner database, we need to either create a new NVIDIA bug or update an existing NVIDIA bug. No Bug ID is required to use the API for creating a new bug, as IDs are assigned automatically to bugs upon creation. Using the API to update an existing bug, however, requires that the bug's NVIDIA Bug ID be passed in as a parameter.

The Google Partner Bugs database does not have a field that stores the corresponding NVIDIA bug ID. We could not control the fields that are stored in the Google database. We engineered a way to obtain the mapping between a Google Partner Bug and the corresponding NVIDIA Bug from the NVIDIA database.

##### **4.2.4.1.1 Mapping Google Bug IDs for Legacy NVIDIA Bugs**

The NVIDIA database did not store Google Bug IDs in any field. While investigating legacy bugs that had been manually synced using the copy-paste system described in Section 2.3.1, we discovered that the NVIDIA engineers had been denoting the Google Partner Bug ID in the *Synopsis* field of the NVIDIA Bug. For example, the Google Bug 123, with synopsis “This is a test bug”, would be stored in the NVIDIA database with the synopsis “[Google Partner Bug 123] This is a test bug”. An example of this can be seen in Figure 6.

**NVBugs** 4.20.2 SC Division: Software Views Report a Bug

**Bug 200076276** Save & Send Clone Instance... Watchlist... Transfer To

Comments Module Info Statuses Dates People Versions Source Control Info Attachments Instance Detail Audit trail Expand ALL sections

Created: 02/02/2015 03:20 AM Fixed: Days Open: 0  
Modified: 02/02/2015 03:20 AM Closed: ID: 200076276

\* Synopsis: [T124/Google/AIO/Kitty][Google Partner Bug 36230] Kitty: factory branch 5772.B stops to build new image, even new CL is merged

Description: [edit](#)

Reported by benjaminl@nvidia.com,  
\*\*\*\*Please do NOT delete Restrict-View-Google label\*\*\*\*

Chrome OS Version : 5772.B  
Type of hardware : Kitty

Please specify Area-\* of the system to which this bug/feature applies (add the label below).

Please help to check why builder stops to build when new CL is merged. Thanks.

**Figure 6: Sample mapping of NVIDIA Bug to Google Bug using *Synopsis* field**  
Reference: [53]

This schema was not strict, however. We found instances of irregular capitalization for the “Google Partner Bug” prefix in the *Synopsis*. This led us to believe that relying on the *Synopsis* field was not a reliable manner of mapping bug IDs.

#### 4.2.4.1.2 Querying NVBugs Using the Internal API

The Google Bug ID being contained in the *Synopsis* field presented another issue for our system. The *Synopsis* field held other text alongside the Google Bug ID; querying for a field with “mixed” values would require a wildcard search. A wildcard search uses the symbols ‘?’ and ‘\*’ as placeholders for individual characters or sequences of characters, respectively. Wildcard searches are used to increase the matches that the search generates. We used the ‘\*’ character to match the text in the *Synopsis* field that surrounded the Google Bug ID. A search for “\*[Google Partner Bug 123]\*” would return all the NVIDIA bugs that had any text surrounding the “[Google Partner Bug 123]” text in the *Synopsis* field.

We observed that wildcard searches on the database were time intensive and unreliable. Rudimentary testing showed that the majority of the queries timed out, which would reduce the reliability of the synchronization system.

#### 4.2.4.1.3 Querying NVBugs Using the Partner Integration API

When investigating the Partner Integration API (Section 3.1.1.1.2), we found that the Partner Integration API exposed fields meant to hold Partner Bug data, such as *Customer Synopsis*,

*CustomerBugID*, *Customer Status*, etc. We utilized the *CustomerBugID* field to hold the Google Bug ID, allowing us to define a mapping between Google Bugs and NVIDIA Bugs.

When a new bug is created in the Google Partner database, we create a new bug in the NVIDIA database using the *SavePartnerBug* function call. We ensure that we set the Google Bug ID as the value for the *CustomerBugID* tag in the Extensible Markup Language (XML) (Section 3.6.1) parameter to this function. We also ensure that we set the *Customer* field to “Google Chrome Bug Sync”. Since there is no bug in the NVIDIA database with the combination of the “Google Chrome Bug Sync” as its *Customer* and the specific Google Bug ID as its *CustomerBugID*, the Partner Integration API understands that this is the creation of a new NVIDIA bug.

When there are updates for a bug on the Google Partner database, we provide the Google Bug's ID number as the value for the *CustomerBugID* tag in the XML parameter to the *SavePartnerBug* function call. The Partner Integration API takes care of synchronizing those updates to the NVIDIA bug that has the same value for its *CustomerBugID*.

#### **4.2.4.2 Advantages of the Partner Integration API**

When using the Partner Integration API, we do not have to query the database using the Google Bug ID to find the matching NVIDIA bug. Using the Partner Integration API removes the potential for queries timing out. We observed no timeouts using this method, which vastly improved our stability from using the Internal SOAP API.

The other advantage of using the Partner Integration API is a reduction in the time needed to update an existing NVIDIA bug. We make a single *SavePartnerBug* API function call, as opposed to two function calls (*GetBugs* followed by *SaveBug*) that we had to make using the Internal SOAP API.

Using the *CustomerBugID* field to store the Google Bug ID allowed us to define a mapping schema that ensures a unique mapping of individual bugs between the two databases. We could not have used the *CustomerBugID* field with the Internal SOAP API, since the Internal SOAP API does not expose the field for interaction.

The Partner Integration API offered an elegant solution to the problem of uniquely mapping Google Bugs to NVIDIA Bugs. We decided to use the Partner Integration API to implement our bug synchronization system's interaction with the NVBugs database.

#### 4.2.5 Tradeoffs around new Mapping Schema

We decided to define a new schema of uniquely mapping Google Partner Bugs to NVBugs, by using the *CustomerBugID* field within an NVIDIA bug to store the corresponding Google Partner Bug ID and using the *Customer* field to store the “Google Chrome Bug Sync” partner name. The team made tradeoffs defining a new schema, which are explored in this section.

##### 4.2.5.1 Legacy Bugs

Legacy bugs that were already in the NVIDIA database did not have the Google Bug ID in their *CustomerBugID* field. Our system would not recognize that a Google Bug (Bug goog-A) mapped to a legacy NVIDIA Bug (Bug nv-A). It would create a new NVIDIA Bug (Bug nv-B) and map that to Bug goog-A. However, Bug nv-A and Bug nv-B would be duplicates of one another. This would lead to time spent manually finding and fixing duplicate bugs, and could also interfere with the bug statistics. We had two options:

- We could manually fix the legacy bugs to adopt the new schema before starting the service.
- We could include some logic in the service to only synchronize Google bugs that were created after a certain date. This date would probably be the date when our service would go live.

Our mentors informed us that there were fewer than 100 legacy bugs, so manually fixing them to adopt our new schema was a possibility. The other option of including code to only synchronize bugs after a certain date would reduce the flexibility of the service. The NVIDIA Project Managers working with the ChromeOS team decided to manually fix the bugs to adopt the new schema.

The other issue for legacy bugs that we encountered was that a single NVIDIA bug could map to multiple Google bugs. For this case, the Project Managers made a decision on which single Google bug they would like to sync with, and entered that Google Bug’s ID in the *CustomerBugID* field.

##### 4.2.5.2 Overwriting the *CustomerBugID* field via the NVBugs Web Interface

The *CustomerBugID* field is modifiable via the NVBugs web interface. If a user modified the value contained in that field, it would change which Google Bug that particular NVBug mapped to. There are three cases that could arise.

#### 4.2.5.2.1 Case 1: Changing to an invalid or nonexistent Google Bug ID

If a user were to change the *CustomerBugID* field's value to something that is not a valid and existing Google Bug ID, our system would no longer find a mapping between a particular NVIDIA bug (Bug nv-a) and the original Google counterpart when updates would next be encountered on the Google side. In this case, the system would simply create a new NVIDIA bug (Bug nv-b). An administrator could easily fix the mistake by marking Bug nv-b as a duplicate of Bug nv-a, and fixing the *CustomerBugID* of Bug nv-a to ensure that all future updates from the Google side were synchronized to Bug nv-a. The administrator would also remove the *CustomerBugID* for Bug nv-b. An NVIDIA user would still be able to see all updates that were made on the Google side for that specific issue between Bug nv-a and the duplicate Bug nv-b.

#### 4.2.5.2.2 Case 2: Changing to another valid Google Bug ID

The second case is if the *CustomerBugID* field for NVIDIA Bug (Bug nv-a) was changed from the Bug ID for one Google Bug (Bug goog-A) to another Google Bug (Bug goog-B). We will assume that Bug goog-B did not already map to another NVIDIA Bug, since that is covered in Section 4.2.5.2.3.

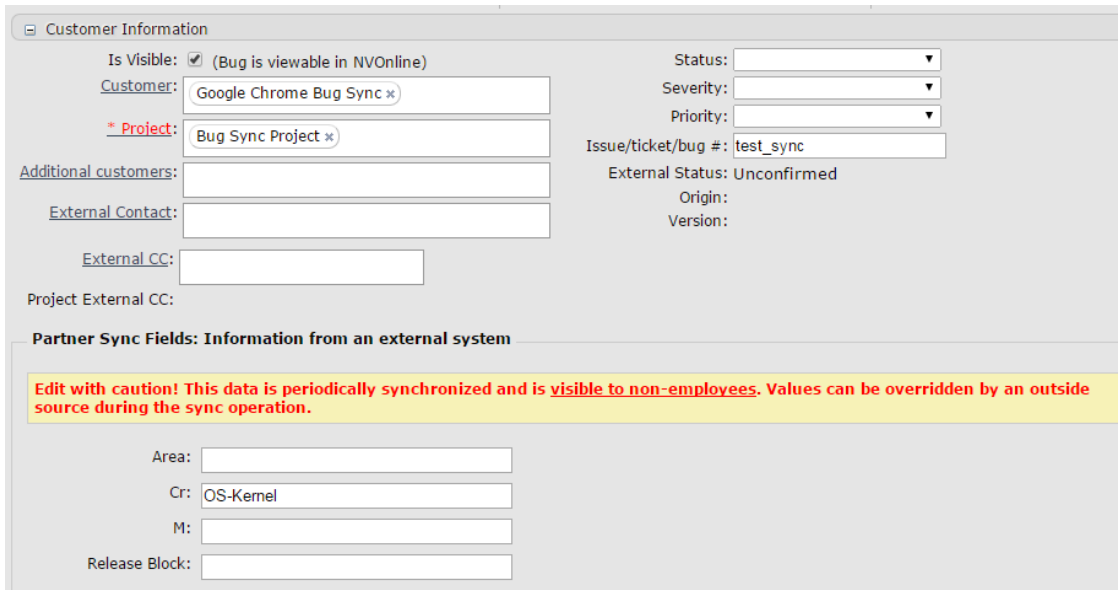
Bug nv-a would already have some updates synchronized for Bug goog-a. When the change is made to the *CustomerBugID* field, Bug nv-a will start synchronizing with Bug goog-b. This will lead to a state where Bug nv-a will have updates from two different Google Bugs.

#### 4.2.5.2.3 Case 3: Changing to another valid Google Bug ID that already maps to an NVIDIA Bug

The final case is if a NVIDIA Bug (Bug nv-a) maps to a Google Bug (Bug goog-a), and another NVIDIA Bug (Bug nv-b) is modified such that its *CustomerBugID* field contains the Bug ID for Google Bug goog-a. Both Bug nv-a and Bug nv-b now map to Bug goog-a. The *GetSyncAudit* function will behave inconsistently when this case is encountered. Both Bug nv-a and Bug nv-b are acceptable results for the query “when was an NVIDIA bug with a *CustomerBugID* that is equal to the Google Bug ID for Bug goog-a last synced to?”, and the *GetSyncAudit* function does not document which of the two bugs it will return in this case. This inconsistency in the behavior of *GetSyncAudit* could lead to undocumented behavior for our service as well.

#### 4.2.5.2.4 Handling Cases 2 and 3

Both Case 2 and Case 3 lead to unwanted behavior in the system. However, modifying the *CustomerBugID* field is not something the Project Managers at NVIDIA do as a matter of routine. We were informed that the engineers who modify bugs also do not commonly modify any of the fields in the “Customer Information” section (the fields available in the “Customer Information” section can be viewed in Figure 7). Since the *CustomerBugID* field is not commonly modified, it was acceptable for the purposes of our project to leave these two cases as documented vulnerabilities.



The screenshot shows a web form titled "Customer Information". It contains several input fields and dropdown menus. The "Customer" field is filled with "Google Chrome Bug Sync". The "Project" field is filled with "Bug Sync Project". The "Status" field is a dropdown menu. The "Severity" field is a dropdown menu. The "Priority" field is a dropdown menu. The "Issue/ticket/bug #" field is filled with "test\_sync". The "External Status" field is filled with "Unconfirmed". The "Origin" field is empty. The "Version" field is empty. There are also fields for "Additional customers", "External Contact", "External CC", and "Project External CC". Below these fields is a section titled "Partner Sync Fields: Information from an external system". This section contains a warning message: "Edit with caution! This data is periodically synchronized and is visible to non-employees. Values can be overridden by an outside source during the sync operation." Below the warning message are fields for "Area", "Cr: OS-Kernel", "M", and "Release Block".

Figure 7: Customer Information Section for an NVIDIA Bug  
Reference: [52]

## 4.3 Logging

We used the Python *logging* library to log messages that would be relevant to tracing the flow of execution of the program and would be helpful for debugging unexpected errors. An overview of the different predefined severity levels provided by the *logging* module is provided in Section 3.7.2.

### 4.3.1 The Logger Hierarchy

The *logging* module organizes all loggers in a program in a hierarchy [84]. All loggers descend from a default "root" logger. Each implemented logger will pass on all events it receives to its parent logger [25]. While the *logging* module allows the building of complex logger hierarchies, we did not use

this functionality. Each file for our service has one logger object which descends directly from the root logger. The logger object in each file receives events from our program and logs the events.

All logger objects have to be configured with a severity level [25]. From all events that are received, only the events whose severity is either equal to or higher than the logger's configured severity will be saved.

#### **4.3.1.1 Custom Logging Level**

While testing the Logger module, we observed that the Suds library generated an error if the logging level was configured at the DEBUG level. This bug has been reported on the Suds GitHub repository [23]. The version of Suds we used did not have a fix for this bug [34].

We wanted to have the capability to log events below the informational level. In case the service encountered an unexpected error, it would be useful to have a log with extra data that could be used for debugging what needed to be fixed. However, this level of verbosity would be too detailed for the INFO log. We decided to use the *logging* module's capability of defining custom logging levels and defined a VERBOSE log level with a numerical value of 15. We used this VERBOSE level to log all of our debug events.

#### **4.3.2 Classes and Objects**

There are three classes of objects from the *logging* module that were of interest to us: Loggers, Handlers, and Formatters.

##### **4.3.2.1 Loggers**

Logger objects provided the interface between our service and the *logging* module. Python's logging facility is set up in a manner such that a Logger object is never instantiated by the application code. Instead, a reference to a Logger object is obtained by a module-level call to the *logging.getLogger(name)* function [67]. Different Logger objects can be instantiated by specifying different values to the name parameter in the function call. For the purposes of our service, we used this function in the following manner:

```
logging.getLogger(__name__)
```

**Code Fragment 7: Obtaining a logger object**

The `__name__` parameter is a special Python construct that parses the name of the module that it is used in. This ensured that the logger objects we created in the separate files had the following properties:

- They were distinct objects, since no two modules can be named the same.
- They were tied to the module that they were logging events from.

#### 4.3.2.2 *Handlers*

Handler objects take care of dispatching log events from the application to the Handler's configured destination. While determining the location to dispatch events to, Handlers also take the level of the log event into consideration (Section 3.7.2). Multiple Handlers can be added to a singular Logger object, allowing for a single log event to be dispatched to multiple locations [68].

We used Handlers to log events of different severity levels to different files. We did this to reduce noise in the log files while still maintaining appropriate verbosity in case an unexpected error occurred.

There are different types of handlers provided in the *logging* module [68]. We used the *RotatingFileHandler* class. This class supports the rotation of log files: when the log file is approaching its maximum size, it can be rolled over into a backup file (with an index number appended to its name) and logging continues in a fresh copy of the log file.

We used this class for three reasons:

- It allowed the log files to be opened in the "append" mode, which allowed us to keep logging to the same file on consecutive runs of the service.
- It allowed us to specify the maximum size for each log file. When the file approached the maximum size, the *RotatingFileHandler* rolled the file over and started using a new file for Logging. This meant that we could define the size of the logs depending on the space constraints of our deployment environment.
- It allowed us to specify whether we wanted to keep the previous log file when a rollover occurred. This allowed us to preserve historical log data, if desired.

We made five different handlers for the logs:

- a *verbose* handler that logged all events from the VERBOSE level onwards to the file "verbose.log"
- an *info* handler that logged all events from the INFO level onwards to the file "info.log"

- a *warning* handler that logged all events from the WARNING level onwards to the file "warning.log"
- an *error* handler that logged all events from the ERROR level onwards to the file "error.log"
- a *critical* handler that logged all events at the CRITICAL level to the file "critical.log"

We created a "logs" folder to hold each of these log files.

#### 4.3.2.3 Formatters

Formatter objects format the contents of the log events [67]. We used a Formatter object to print detailed time information for each log event. We configured the Formatter in the following manner:

```
%(asctime)s - %(name)s - %(levelname)s - %(message)s
```

#### Code Fragment 8: Configuration of Logging Formatter

The pieces of this message are as follows:

- *asctime*: a function from the Python *time* library. Converts the local time into a user-readable string. For example, 'Sun Jun 20 23:21:05 1993' [72].
- *name*: the name of the Logger object that is printing the event.
- *levelname*: the severity level of the event.
- *message*: the message pertaining to the event.

Using this format, a sample log message looks like:

```
2015-02-23 09:21:46,231 - __main__ - INFO - Obtained NVBugs API client
successfully.
```

#### Code Fragment 9: Sample Log Message

### 4.3.3 Configuration

Logging configuration in Python is at the application level. We configured our logging setup in two files: *logging.yaml* and *setup\_logging.py*.

#### 4.3.3.1 *Logging.yaml*

We used the YAML data format to create a file that contained the configuration settings for the *logging* module. This allowed us to have the entire logger hierarchy structure and the specification of the logging levels outside of the Python modules. Using a separate configuration file made for a logical separation of the logging configuration from the application code, and allowed for flexibility in toggling the configuration without having to change the application code.

We specified the Formatter object's configuration in the YAML file. We created the different Handler objects with the desired severity configuration; one Handler for each of the severity levels of VERBOSE, INFO, WARNING, ERROR, and CRITICAL. We connected each Handler to the respective file that we wanted the log messages of those severity to be printed in, and specified the maximum size of each log file to be 0.5 megabytes.

We also set each Handler to keep one historical record of each log file. During a file rollover for *info.log*, a file *info.log.1* would be created and INFO level messages would continue being logged to a fresh file named *info.log*. When the new *info.log* would reach the maximum size allowed, the old *info.log.1* would be deleted, *info.log* would be moved into *info.log.1*, and a fresh copy of *info.log* would be used for logging. Each of the log files would behave in the same manner.

#### 4.3.3.2 *Setup\_logging.py*

We read the YAML file in the *setup\_logging.py* module, and used the values from the configuration file to set up the *logging* module. To actually use the Logging facility, we had to call the *setup\_logging* function in our main *google\_partner\_sync.py* file. We then created a global logger object for each file that we wished to log in:

```
import logging

logger = logging.getLogger(__name__)
```

**Code Fragment 10: Setup for a logger object**

We could then log messages in the following manner:

```
logger.severity(message)
```

The *severity* is the level at which we want this event to be treated. The *message* is the information that we want to print when this event occurs. For example, an informational message can be logged as:

```
logger.info('This is an informational message.')
```

We created the custom VERBOSE level logger in this file. We included functionality that registered the VERBOSE level logger with the Python Logging facility, which allowed us to make VERBOSE level calls in a form emulating that of the predefined levels:

```
logger.verbose(message)
```

#### **4.3.4 Logging Tradeoffs**

We made decisions about what severity levels would be assigned to the different messages from the program.

##### **4.3.4.1 VERBOSE**

We logged messages relating to the function that the script was performing at particular times. This would help with debugging the script if any unexpected errors were encountered.

We logged the time at which each Google bug was last updated on the Google side, and the time the corresponding NVIDIA bug was last synchronized. This would help with debugging any issues with the *GetSyncAudit* API function.

We logged all the XML generated for synchronizing each Google bug with the corresponding NVIDIA bug. This would help with debugging any issues with the *SavePartnerBug* API function.

##### **4.3.4.2 INFO**

At this level, we logged informational messages relating to the overall functionality of the script. This log is intended to fill an administrator's needs, as long as the script is performing as expected.

We logged the number of Google bugs fetched from the *chrome-os-partner* project, and the size of the subset of Google bugs that actually had updates that required synchronization. We also logged the Google Bug IDs for any Google bugs that we were unable to successfully synchronize. Finally, the

time that the script took to complete the entire synchronization operation was also logged at the INFO level.

#### **4.3.4.3 WARNING**

We logged any issues we encountered while scraping the HTML elements for any Google bug. These included issues like an incorrect page being served, the HTML the service encountered was different from what was expected, or Google's server denying our HTTP requests. These are all issues that are either specific to a single bug's HTML page structure, or are issues that are temporary and would get resolved on re-running the script.

We logged any failures encountered when using the *GetSyncAudit* method and the *SavePartnerBug* method. These were both logged at the WARNING level, as API call failures could be due a wide variety. A couple of potential failures for our service could be network connectivity issues or server outage. The majority of scenarios leading to an API failure should be resolved the next time the script executes.

#### **4.3.4.4 ERROR**

We logged any errors encountered when trying to obtain a client to interact with the NVBugs Partner Integration API. Failing to obtain a client causes the script to abort the current run.

We logged any authentication failures encountered when making a request to the Google servers. This could happen if the "bug-sync@nvidia.com" Google account's password changes.

We also logged any errors encountered on making the *SavePartnerBug* API call. The API will generate errors if any issues were encountered with malformed XML or with the Suds library. These errors will not be fixed by re-running the script; they need an administrator to ensure that the code still conforms to the API's WSDL (Section 3.6.3). We flag API errors at the ERROR level to ensure that an administrator views the messages.

#### **4.3.4.5 CRITICAL**

We logged any authentication failures encountered when trying to connect to NVBugs, since this would need immediate attention from the system administrator. We also logged any errors encountered around decrypting passwords as critical for similar reasons.

## 4.4 Deployment

The bug synchronization service went through many stages of approval and setup before being able to deploy on the production NVBugs database. During this time, the service moved between several different machines.

### 4.4.1 Development

We developed the bug synchronization service on a Linux computer running the Ubuntu 12.04 LTS operating system [17]. This computer was used for development by engineers on the ChromeOS team, and was provisioned to us for our project at NVIDIA. We had sole access to the computer, and had permissions to change the environment setup as needed. Each member of the team had a user account on this computer, and used it as their workspace for the project. All members of the team could access their user space on the Linux computer at any time using Secure Shell (SSH) from the Windows computers at their desks (for more information on SSH, see [86]).

### 4.4.2 Testing

We were provisioned a Virtual Machine (VM) for testing once the service reached a stable point. This VM used the CentOS – Release 6.5 operating system [83]. We were also provisioned a network mounted folder with 5 GB of memory in our home directories on the VM. We added the code for our synchronization service into the folder in our home directory by cloning our Git repository (for more information on Git repositories, see [22]).

Our user accounts on the testing VM were not given any “sudo” permissions [94]. *Sudo* is a program that allows system administrators to specify the level of authority each user has. Without any *sudo* privileges given to our user account, we were unable to access or modify files in certain directories, which limited our ability to modify the VM’s environment. We instead used a directory setup that mimicked the structure of the Linux directories “/usr/bin/” and “/usr/lib/” in our home directory (for more information on the standard structure of Linux directories, see [51]). The addition of “bin” and “lib” directories in our home directory allowed us to install needed applications such as Git – Version 1.8 and Python – Version 2.7 to use locally, without needing access to the “/usr/” directory level which required *sudo* permissions.

We used the testing VM to run our service, both manually and on pre-configured intervals for testing using the *Cron* service (Section 3.4.1.1). We ran the service on all open and closed bugs in the

ChromeOS project in the Google Project Hosting system. There were 1161 bugs fitting this criteria, which gave us the opportunity to find edge cases that we had not considered.

#### 4.4.3 Deploying to Production

It was important to us and the NVIDIA ChromeOS team to have the bug synchronization service running live for at least a week before our project was done so that we could be present to help with any unexpected problems. The deployment process involved passing the bug synchronization service to the team that would be maintaining it once we left, and configuring the service to run on production NVBugs database instead of the development database. Two Virtual Machines (VMs) were provisioned: a development VM, and a production VM.

The development VM was created as a staging environment where we could set up our working service to be moved to the production VM by NVIDIA's MIS- Farm Support team. On the development VM we were given *sudo* permissions, which allowed us to set up the service in its own directory: "GoogleChromeSync" in the "/usr/local/" space. We configured Python – Version 2.7 with the additional modules we needed as dependencies in "/usr/bin/" as an alternate install of Python. We needed to make our version of Python an alternate install due to the CentOS operating system relying on Python – Version 2.6 to run properly. With our version as an alternate install, we were able to maintain the install of Python – Version 2.6 and still run our service using Python – Version 2.7 by using the command "python2.7" instead of the typical "python" command.

After successfully configuring the bug synchronization service on the development VM, we wrote a set of detailed instructions for how to move the code from the development VM to the production VM. We simplified the transfer of the project from the development VM to the production VM so that the administrator performing the transfer would only need to copy files and encrypt the password on their machine. We also made a configuration file (as observed in Appendix F) of constants which required the administrator to change 'DEV' to 'PROD' in order to change the various URLs and accounts being used by the bug synchronization service from development to production. For the full set of instructions, see Appendix G. Leandro Awa, a member of the NVIDIA MIS- Farm Support team followed the instructions to perform the transfer from the development VM to the production VM.

In addition to the initial deployment instructions, we provided a detailed set of documentation to ensure easy maintenance of the synchronization service. The complete documentation can be seen in Appendix H.

## 4.5 Statistics

Our synchronization service was built to expedite the tasks of the Project Managers (PMs). The PM's jobs also consisted of tracking the progress of the ChromeOS development team, as explored in Section 3.8.1. The PMs track the progress of their team through monitoring bug data. The more bugs the ChromeOS team fixes, the closer the team is to shipping a product.

Before our service, the PMs manually analyzed the data, explained in Section 3.8.1. To help expedite the task of generating bug progress reports, we developed a statistics generator. This tool automates the process of generating bug trends.

### 4.5.1 Preparing the Environment

We developed the statistics program in Microsoft Excel. Microsoft Excel helps streamline the chart generation, and provides an easy-to-use spreadsheet for tabulating and visualizing data. Microsoft Excel also provides access to Virtual Basic for Applications (VBA) (Section 3.8.1), which helped with the automation of generating SQL queries.

We used the Gremlins database to generate the relevant data for the statistics program. The Gremlins database is described in detail in Section 3.8.2.2. We connected to the Gremlins database. The connection to Gremlins is called the “NVSQ113/SQLPROD113 Database” connection. This connection allowed us to run SQL queries against the Gremlins database, generating tables of relevant bug information.

### 4.5.2 Connecting to the Gremlins Database

To connect to the database, we employed Microsoft Excel's SQL Connection Wizard. For further information on SQL Connection Wizard, see Microsoft's Documentation [49]. This tool allowed us to create a connection to a SQL Database using a target URL (NVSQ113/SQLPROD113) and valid credentials supplied through the user's desktop. Each time we wanted to run a different query on the database, we needed to create a new connection. Each time a connection is created, we named it appropriately. We created the following connections:

- *AllBugsOpen*: For the bug velocity open bugs
- *AllBugsNew*: For the bug velocity new bugs
- *AllBugsFixed*: For the bug velocity fixed bugs
- *TrendBugs*: For the bug trends per module

- *EngineerBugs*: For the bug trends per engineer

Each of these connections enabled a single SQL query to be run. The SQL query is unable to dynamically change. The Connection Wizard does not allow referencing outside values. If the user wanted to change a query's values, the user would need to go into the connection, and manually change the SQL query to the appropriate values. We automated this process through the use of Microsoft's Visual Basic for Applications.

#### 4.5.3 Automating SQL Connection with Visual Basic for Applications

Whenever the values of a SQL query would change, such as the *CustomerProject* field or the date used, the user would need to open the relevant SQL Connection and change the SQL query field.

We were able to automate the process of opening the connection, generating the SQL query, and refreshing the query by using Visual Basic for Applications (VBA).

An example of the code we used to generate the SQL can be found in Code Fragment 11. This code opens the SQL's *.OLEDB* connection, giving access to the SQL Connection's *CommandText* field. The *CommandText* field is where the SQL query is run from. We then generated the SQL query (explained in Section 4.5.4), and closed the *.OLEDB* connection, saving the *CommandText* field. The connection was then refreshed, and the Microsoft Excel spreadsheet would send the SQL query to the Gremlins database. The Gremlins database would return the requested bug data to the Microsoft Excel spreadsheet.

```
'Generate a table of bug trend by module information
'Lists only the top N modules as of lastReport
Sub GenTrendNumTable(projName As String, modName As String,
    numReps As Integer, lastReport)
    With ActiveWorkbook.Connections("NVSQL113_SQLPROD113
        ModuleTrends").OLEDBConnection
        .CommandText = GetTopTrendsString(projName, numReps,
            lastReport)
    End With
    ActiveWorkbook.Connections("NVSQL113_SQLPROD113 ModuleTrends").Refresh
End Sub
```

**Code Fragment 11: Connection Code**

#### 4.5.4 Automating SQL Generation with Visual Basic for Applications

Once we were able to run SQL commands programmatically through VBA, we needed to generate the SQL commands that would retrieve the relevant data from the Gremlins Database. We built a single function, *RefreshAllData*, to refresh all the queries we would create. *RefreshAllData* would call three sub-functions:

- *GenAllBugsTable*
- *GenTrendTable*
- *GenEngineerTable*

Each of the three sub-functions would generate their own SQL query, and then insert the generated query into their respective connections. The spreadsheet would then be refreshed, as described in Section 4.5.3.

##### 4.5.4.1 *GenAllBugsTable*

The *GenAllBugsTable* sub-query generates the data that the Bug Velocity Chart uses, as seen in Figure 2. The sub-query generates three different bug types – Open, New, and Fixed – by executing the following pseudocode loop:

- For Each Week in the requested data:
  - For each bug:
    - Find the changes to the *BugAction* field for that week
    - Add that bug to either the Open, New, or Fixed bugs based on the changes
  - Count the number of bugs in each list of Open, New, or Fixed bugs

The *GenAllBugsTable* sub-query iterates over data for the past 15 weeks. The sub-query checks each bug's *BugAction* field at the end of each week to determine if it was open, new, or fixed. There were three scenarios for determining which list the bug would be placed under:

- If, at the end of the week, the *BugAction* was equal to one of the options below, the bug would be considered Open:
  - Dev - Open - To fix
  - Dev - Open - To Triage
  - QA - Open - Provide more detail
  - QA - Open - Request details from customer

- If, during the week, the *BugAction* changed from one of the options below to any *BugAction*, the bug would be considered New:
  - [blank]
  - QA - Verify new bug
- If, during the week, the *BugAction* changed from one of the options in the above Open values to one of the options below, the bug would be considered Fixed. (If the bug was later changed back to one of the above Open values, it would not be counted as Fixed.)
  - Dev - Open - Verify to Close
  - QA - Open - Verify to close
  - Dev - Closed - Unverified
  - Dev - Closed - Verified
  - QA - Closed - Unverified
  - QA - Closed - Verified

#### 4.5.4.2 *GenTrendTable*

The *GenTrendTable* sub-query generates the data that the Bug Trends per Module Chart uses, as seen in Figure 3. The sub-query generates the number of open bugs per module by executing the following pseudocode loop:

- For each week in the requested data:
  - For each module:
    - For each bug:
      - Find the most recent change to the *BugAction* field for that week
      - Add that bug to the list of Open bugs
    - Count the number of bugs in the list of Open bugs
  - Count the number of Open bugs for each module

The *GenTrendTable* sub-query iterates over data for the past 15 weeks. The sub-query checks each bug's *BugAction* field at the end of each week to determine if it was Open. If the *BugAction* field matched one of the values below, it would be considered Open:

- Dev - Open - To fix
- Dev - Open - To Triage

- QA - Open - Provide more detail
- QA - Open - Request details from customer

#### 4.5.4.3 *GenEngineerTable*

The *GenEngineerTable* sub-query generates the data that the Bug Trends per Engineer Chart uses, as seen in Figure 4. The query generates the number of open bugs per engineer by executing the following pseudocode loop:

- For each engineer in the requested data:
  - For each bug:
    - Find the most recent change to the *BugAction* field for that week
    - Add that bug to the list of Open bugs
  - Count the number of bugs in the list of Open bugs
- Count the number of Open bugs for each Engineer

The *GenEngineerTable* sub-query runs for only the current week. The query checks each bug's current *BugAction* field if it is Open. If the *BugAction* field matched one of the values below, it would be considered Open:

- Dev - Open - To fix
- Dev - Open - To Triage
- QA - Open - Provide more detail
- QA - Open - Request details from customer

#### 4.5.5 Creating a User Interface for the Project Managers

We created a User Interface (UI) in Microsoft Excel to simplify the use of the statistics tool for the Project Managers (PMs).

Figure 8 shows the UI for the tool. The PMs can use the various inputs, described below, to modify their search query. They can then trigger the data retrieval by pressing the "Refresh" button.

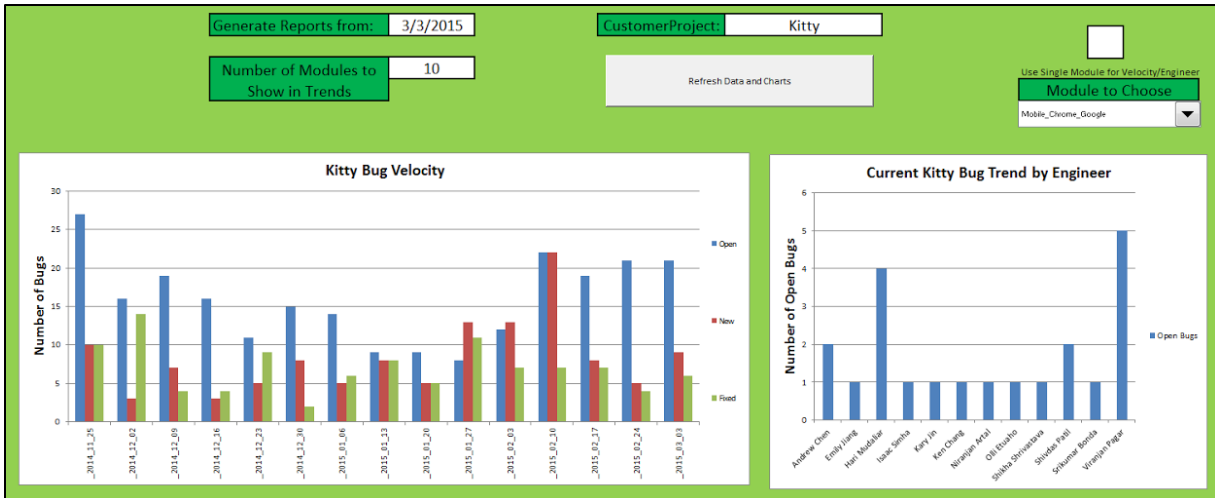


Figure 8: The User Interface of the Statistics Tool

The UI allows for the following fields to be changed for the search query:

- “Generate Reports From” Date

This is the date that the charts will be generated from. The SQL queries will iterate over data going 15 weeks in the past from this date.

- “Number of Top Modules to Show in Trends” Number

This is the number of modules that the Trends per Module chart will display. The SQL query will only request data for this number of trends. The modules selected will be determined based on the number of open bugs for each module, in the order of most to least. The rest of the modules will be ignored.

- “CustomerProject” Text

This is the value that will be searched against the Gremlins database. The SQL queries will only request bugs that have this value in their *CustomerProject* fields.

- “Refresh Data and Charts” Button

This is the button the user of the tool can use to start the querying process. This will refresh all the data present in the program, generating new charts using the inputs provided by the user in the Excel spreadsheet.

- “Use a Single Module” Check Box

This is the check box to toggle whether or not to search using only a single module. The single module to use is determined by the “Use a Single Module” drop down list.

- “Use a Single Module” Drop Down List

This is a list where the user using the tool can select the individual module to search by. The list refreshes when the “Use a Single Module” Check Box is checked, or the “Refresh Data and Charts” Button is pressed. The list will not refresh on the “Use a Single Module” Check Box being unchecked.

Below the input fields, the UI displays the three charts generated from the SQL queries. The generated charts can be seen in Figure 9. Information on these charts can be found in Section 3.8.1. These charts will be refreshed after the “Refresh Data and Charts” Button is pressed.

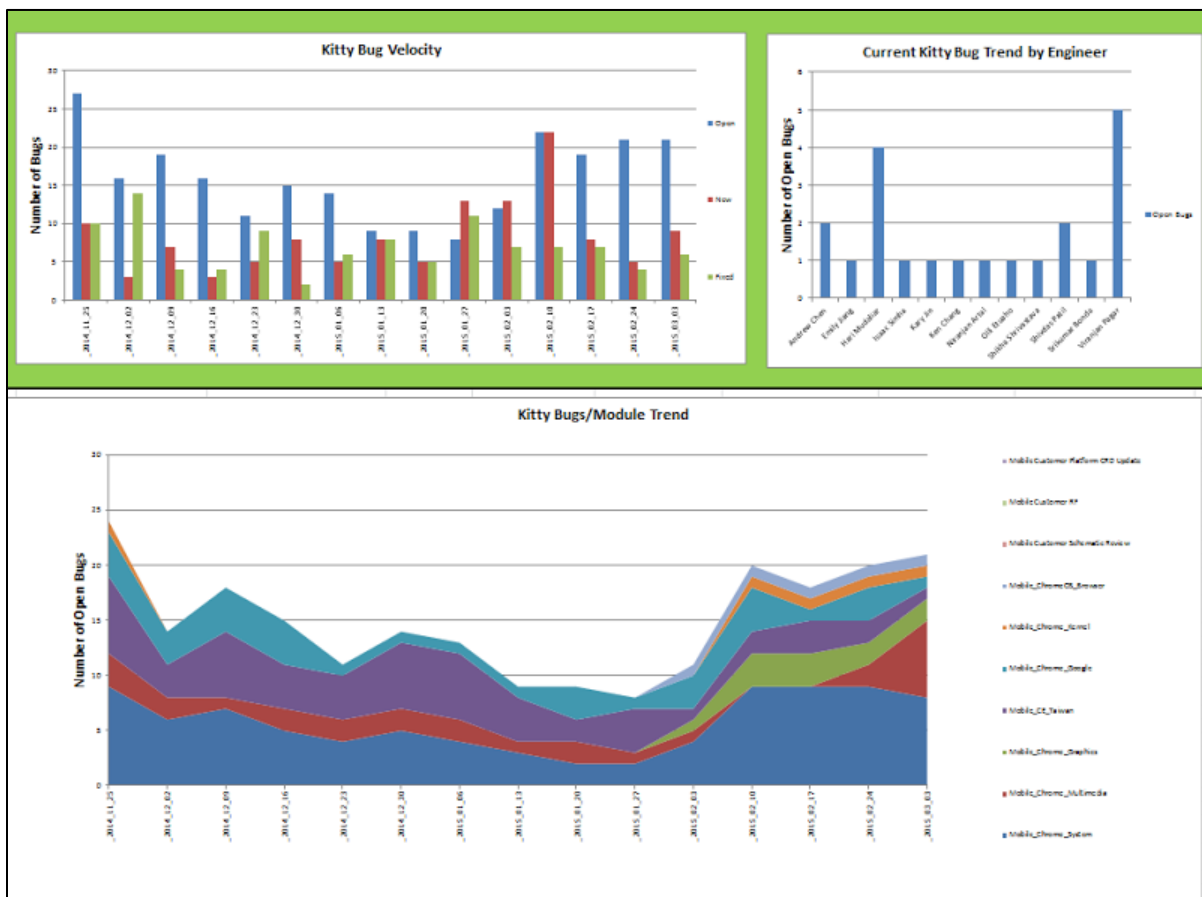


Figure 9: UI for Generated Charts

## 5 Results

We completed deliverables for both the Bug Synchronization Service and Bug Statistics Tool. Both deliverables will reduce the time the ChromeOS Project Managers spend on manual data entry and tabulation. Both deliverables have the possibility for extension by future projects.

### 5.1 Bug Synchronization Service

The Bug Synchronization Service was moved to a production server on February 27th, 2015, one week before our project ended. The service successfully caught all updates for NVIDIA-owned Google Bugs and synchronized those updates to NVBugs, as expected. We helped NVIDIA engineers on the TechOps team and MIS-Farm Support team deploy the service and set up a monitoring system. The monitoring system sends alerts to a distributed mailing list whenever the service is not running successfully. Once our service was running on the production server, we accumulated a number of statistics based on its performance, as seen in Table 2.

**Table 2: Bug Synchronization Performance Statistics**

Average number of bugs retrieved	29.46
Average number of bugs needing sync	0.04
Average time to run (seconds)	8.140371

For the full set of data used to calculate the Bug Synchronization Performance Statistics, see Appendix I. The “Average number of bugs retrieved” represents the number of open bugs in the Google Partner database that are assigned to NVIDIA engineers. The “Average number of bugs needing sync” statistic shows how often the service encountered a bug that needed to be updated on the NVIDIA bug tracker. The value, 0.04, showed that every 4 in 100 bugs needed an update, or that less than 1 in 20 synchronization runs encountered any bugs that needed to be updated. The average time to run, 8.140371 seconds, represents the average amount of time in seconds that it took for a complete run of the Bug Synchronization Service, regardless of the number of updates for each run. A runtime of less than 10 seconds exceeded our time requirements for the speed of the service. We decided to run the Bug Synchronization Service every ten minutes, meaning the longest delay for a change on a Google Partner Bug to be reflected in the NVBugs database would be 10 minutes. This delay is much smaller

than when Project Managers would manually copy and paste updates to NVIDIA bugs, as Project Managers often only had time to update the bugs at most once a day.

### **5.1.1 Extending the Bug Synchronization Service**

The Bug Synchronization Service has two main parts: obtaining the data from Google, and updating the NVIDIA bug database. We architected the code for the service in such a way that either data source can be replaced, as long as the correct input and output structures are maintained. This allows for easy extendability of our service.

Our service works with Python dictionaries [65] containing bug information. The dictionaries hold the following fields:

- ID
- Status
- AllLabels
- Description
- Comments
- M
- Modified
- ModifiedTimestamp
- Summary
- Proj
- Owner
- Cr
- OS

If the method to obtain bug information changes, as long as the new method returns a dictionary with those fields, it will work with the code to update the NVIDIA bug database. In addition, specific pieces of information such as constants, URLs, accounts, and passwords have been extracted to a configuration file (Appendix F). NVIDIA could use the code to update the NVIDIA bug database for other partners by changing the code to obtain the bug information, and filling in the Configuration file to match the new partner information.

## 5.2 Bug Statistics Tool

The Bug Statistics tool was given to project managers for review on March 2nd, 2015. We developed two versions of the Bug Statistics tool: a BugAudit version, and a Timestamp version. The BugAudit version checks bug status using the *BugAction* attribute. The *BugAction* attribute is used to determine if the bug is open, fixed, or new. The Timestamp version checks a bug's *CreatedDate*, *FixedDate*, and *ClosedDate* timestamps to determine if a bug is open, fixed, or new.

Both versions of the Bug Statistics tool markedly shortened the time necessary for the Project Managers to collect data from NVBugs. The BugAudit version took an average of 4.5 minutes to generate all the data, while the Timestamp version took an average of 80 seconds to generate all the data.

The three charts that were requested in Section 3.8.1 were generated with both versions of the tool, with accurate data.

### 5.2.1 Extending the Bug Statistics Tool

We worked with the NVBugs development team while producing the Bug Statistics spreadsheets for the Project Managers. Karthik Gopalakrishnan, one of the lead engineers on the Gremlins database development, is very interested in developing a company-wide visualization tool for bug trends. He plans on integrating the SQL logic we developed into his visualization tools, so that any project manager on any team will be able to generate custom charts rapidly. The VBA code will be replaced with his programming language of choice.

Sumanta Chakraborty, one of the Project Managers we worked with, was interested in using data mining on the programmatically-retrieved data. The data would be useful for predicting future milestones for each project, such as when only a certain number of bugs would remain. Other benefits of data mining could include projecting project completion dates based on the trends in bug statuses and the number of engineers assigned to the project. Karthik plans on integrating prediction modeling in the future visualization tools using the SQL logic we developed.

## 6 Conclusion

We developed two services for NVIDIA:

- A Bug Synchronization service that synchronizes bugs for the ChromeOS team from the Google ChromeOS Partner Bugs database to the NVIDIA NVBugs database.
- A Bug Statistics tool that generates bug trends and visualizations for any Customer Project present in the NVIDIA NVBugs database.

These two pieces of functionality were previously accomplished by NVIDIA Project Managers manually manipulating data:

- They would copy and paste bugs between the Google ChromeOS Partner Bugs database and the NVIDIA NVBugs database.
- They would tabulate data from the NVIDIA NVBugs database manually, and then generate the required visualizations from these tables.

The previous methods of manipulating bug data were infrequent, error-prone, and time-intensive. The methods of accomplishing the same tasks that we developed are automated, accurate, and much quicker.

Both of our services have potential for further extensions. The Bug Synchronization service can be extended to establish one-way bug synchronization with any NVIDIA partner. The Bug Statistics tool can be used to develop a company-wide bug trend visualization tool.

Both of our solutions reduce the time Project Managers spend on manual bug data manipulation, thereby helping increase Project Managers' productivity.

## 7 Bibliography

- [1] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.001>
- [2] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.004>
- [3] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.005>
- [4] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.006>
- [5] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.007>
- [6] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.012>
- [7] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.013>
- [8] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MPRAA.013>
- [9] Acer Inc. "Acer Chromebook". Retrieved: March 02, 2015.  
<http://us.acer.com/ac/en/US/content/model/NX.MRDAA.003>
- [10] Advameg Inc. "PGP Frequently Asked Questions Part 1/3". Retrieved: February 7th.  
<http://www.faqs.org/faqs/pgp-faq/part1/>
- [11] Agarwal, Guarav. "Tegra K1 Transforming Chromebooks From the Inside Out". Modified: August 11, 2014. Retrieved: March 02, 2015. <http://blogs.nvidia.com/blog/2014/08/11/tegra-k1-chromebooks/>
- [12] Anders Møller and Michael I. Schwartzbach. Recipe Server WSDL.  
<http://www.brics.dk/ixwt/examples/recipeserver.wsdl>. Retrieved: February 22, 2015.
- [13] "Are CentOS programs compatible with Ubuntu?" Modified: July 19, 2012. Retrieved: February 17, 2015. <http://askubuntu.com/questions/165643/are-centos-programs-compatible-with-ubuntu>

- [14] Audi AG. "Welcome to the World of Audi". Retrieved: March 1, 2015.  
<http://www.audi.com/index.html>
- [15] Box, Don, et al. "Simple object access protocol (SOAP) 1.1." (2000).
- [16] Bray, Tim, et al. "Extensible markup language (XML)." World Wide Web Consortium Recommendation REC-xml-19980210. <http://www.w3.org/TR/1998/REC-xml-19980210> (1998): 16.
- [17] Canonical Ltd. "Meet Ubuntu". Modified: 2015. Retrieved: February 22, 2015.  
<http://www.ubuntu.com/desktop>
- [18] Chinnici, Roberto, et al. "Web services description language (wsdl) version 2.0 part 1: Core language." W3C recommendation 26 (2007): 19.
- [19] Davis, Matthew. "What is The Difference Between CentOS and Ubuntu Server?" Modified: March 25, 2014. Retrieved: February 17th, 2015. <https://www.futurehosting.com/blog/what-is-the-difference-between-centos-and-ubuntu-server/>
- [20] Dürst, Martin, and Asmus Freytag. "Unicode in XML and other Markup Languages". Vol. 20. Unicode Technical Report, 2002.
- [21] Evans, Clark. "YAML: YAML Ain't Markup Language". Retrieved: February 23, 2015.  
<http://yaml.org/>
- [22] Git. "2.1 Git Basics - Getting a Git Repository". Retrieved: February 26, 2015. <http://git-scm.com/book/en/v2/Git-Basics-Getting-a-Git-Repository>
- [23] GitHub, Inc. "Returning NoneType causes an error when trying to write log info to a file". Retrieved: February 23, 2015. <https://github.com/fabrique/python-suds/issues/3>
- [24] GnuPG. "Gnu Privacy Guard". Retrieved: February 7th. <https://www.gnupg.org/>
- [25] Gollakota, Praveen. "How Python logging module works. Shut Up and Ship." Retrieved: February 23, 2015. <http://www.shutupandship.com/2012/02/how-python-logging-module-works.html>
- [26] Google Inc. "About Google". Retrieved: March 1, 2015.  
<https://www.google.com/intl/en/about/>
- [27] Google Inc. "Nexus 9". Retrieved: March 1, 2015. <http://www.google.com/nexus/9/>
- [28] Google Inc. "Chrome Devices". Retrieved: March 2, 2015.  
<https://www.google.com/chrome/devices/>

- [29] Google Inc. "Chromium Issues". Retrieved: March 2, 2015.  
<https://code.google.com/p/chromium/issues/csv?can=2&q=&colspec=ID%20Pri%20M%20Week%20ReleaseBlock%20Cr%20Status%20Owner%20Summary%20OS%20Modified&sort=-pri>
- [30] Google Inc. "Chromium Lists". Retrieved: March 2, 2015.  
<https://code.google.com/p/chromium/issues/list>
- [31] Google Inc. "Chromium". Retrieved: March 2, 2015. <http://www.chromium.org/>
- [32] Google Inc. "Google Project Hosting". Retrieved: March 2, 2015.  
<https://code.google.com/projecthosting/>
- [33] Google Inc. "Meet Chromebook". Retrieved: March 02, 2015.  
<http://www.google.com/chrome/devices/features/>
- [34] Google Project Hosting. "Issue 1100: Tests using suds fail if log level is below info". Retrieved: February 23, 2015. <https://code.google.com/p/robotframework/issues/detail?id=1100>
- [35] Google Project Hosting. "Issue 33624: Mipi TX THS-TRAIL". Retrieved: March 03, 2015. view-source:<https://code.google.com/p/chrome-os-partner/issues/detail?id=33624&q=Comment%3A%22We%20found%20out%20that%20the%20calculations%20in%20the%20driver%22&colspec=ID%20Pri%20M%20ReleaseBlock%20Cr%20Status%20Owner%20Summary>
- [36] Hewlett-Packard Development Company, L.P. "HP Chromebooks." Retrieved: March 1, 2015.  
<http://www8.hp.com/us/en/ads/chromebooks/overview.html>
- [37] Hewlett-Packard Development Company, L.P. "HP Chromebook - 14 - x010nr". Retrieved: March 02, 2015. <http://store.hp.com/webapp/wcs/stores/servlet/us/en/pdp/Laptops/hp-chromebook---14-x010nr-energy-star>
- [38] Hewlett-Packard Development Company, L.P. "HP Chromebook - 14 - x030nr". Retrieved: March 02, 2015. <http://store.hp.com/webapp/wcs/stores/servlet/us/en/pdp/Laptops/hp-chromebook---14-x030nr-energy-star>
- [39] Hewlett-Packard Development Company, L.P. "HP Chromebook - 14 - x040nr". Retrieved: March 02, 2015. <http://store.hp.com/webapp/wcs/stores/servlet/us/en/pdp/Laptops/hp-chromebook---14-x040nr-energy-star>
- [40] Hewlett-Packard Development Company, L.P. "HP Chromebook - 14 - x050nr Touch". Retrieved: March 02, 2015.

<http://store.hp.com/webapp/wcs/stores/servlet/us/en/pdp/Laptops/hp-chromebook---14-x050nr-%28energy-star%29#!&TabName=specs>

- [41] HTC Corporation. "HTC One X+". Retrieved: March 1, 2015.  
<https://www.htc.com/us/smartphones/htc-one-x-plus/>
- [42] ICU User Guide. "Unicode Basics". Retrieved: February 16, 2015. <http://userguide.icu-project.org/unicode>
- [43] Internet Engineering and Task Force. "Common Format and MIME Type for Comma-Separated Values (CSV) Files". Retrieved: March 2, 2015. <http://www.ietf.org/rfc/rfc4180.txt>
- [44] "Internet Message Access Protocol - Version 4rev1". Modified: March 2003. Retrieved: February 08, 2015. <https://tools.ietf.org/html/rfc3501>
- [45] "Internet Protocol Standards". Modified: May 2008. Retrieved: February 8, 2015.  
<https://tools.ietf.org/html/rfc5000>
- [46] Intertwingly. "RSS 20 and Atom 10 Compared". Retrieved: March 2, 2015.  
<http://www.intertwingly.net/wiki/pie/Rss20AndAtom10Compared>
- [47] Lee, Titan. NVIDIA Corporation. "HP Chromebook SW Bug: Breakdown Charts". Retrieved: March 5, 2015. <https://nvbugswb.nvidia.com/NVBugs5/?dvid=3#132239/BUGPRIORITY=:0::0>
- [48] Microsoft Corporation. "About Microsoft". Retrieved: March 1, 2015.  
<http://www.microsoft.com/about/en/us/default.aspx>
- [49] Microsoft Corporation. "SQL Server Import and Export Wizard". Retrieved: March 2, 2015.  
<https://msdn.microsoft.com/en-us/library/ms141209.aspx>
- [50] Microsoft Corporation. "Welcome to the Visual Basic for Applications language reference for Office 2013". Retrieved: February 23, 2015. <https://msdn.microsoft.com/en-us/library/office/gg264383.aspx>
- [51] Natarajan, Ramesh. "Linux Directory Structure (File System Structure) Explained with Examples". Modified: September 8, 2010. Retrieved: February 22, 2015.  
<http://www.thegeekstuff.com/2010/09/linux-file-system-structure/>
- [52] NVIDIA Corporation. "Bug 1390925" Retrieved: March 2, 2015.  
<http://hqdvbugswb03.nvidia.com/NVBugs5/SWBug.aspx?dvid=1&bugID=1390925&cp=&qid=&lp=&cmt=&synopsis=&engineer=&arb=&severity=&module=&backtomywork=&cmtNo=>

- [53] NVIDIA Corporation. "Bug 200076276". Retrieved: February 18, 2015.  
<https://nvbugswb.nvidia.com/NVBugs5/SWBug.aspx?dvid=1&BugID=200076276>
- [54] NVIDIA Corporation. "External Partner Integration". Retrieved: March 2, 2015.  
[https://wiki.nvidia.com/nvbugswiki/index.php/External\\_Partner\\_Integration](https://wiki.nvidia.com/nvbugswiki/index.php/External_Partner_Integration)
- [55] NVIDIA Corporation. "NVBugs SOAP". Retrieved: February 2, 2015.  
[https://wiki.nvidia.com/nvbugswiki/index.php/NVBugs\\_SOAP](https://wiki.nvidia.com/nvbugswiki/index.php/NVBugs_SOAP)
- [56] NVIDIA Corporation. "NVIDIA History". Retrieved: March 1, 2015.  
[http://www.nvidia.com/page/corporate\\_timeline.html](http://www.nvidia.com/page/corporate_timeline.html)
- [57] NVIDIA Corporation. "NVIDIA Partner Network". Retrieved: March 1, 2015.  
<http://www.nvidia.com/object/nvidia-partner-network.html>
- [58] NVIDIA Corporation. "NVIDIA Tegra X1". Retrieved: March 1, 2015.  
<http://www.nvidia.com/object/tegra-x1-processor.html>
- [59] NVIDIA Corporation. "Tegra K1 Next-Gen Mobile Processors". Retrieved: March 2, 2015.  
<http://www.nvidia.com/object/tegra-k1-processor.html>
- [60] NVIDIA Corporation. "The Ultimate Gaming Portable". Retrieved: March 1, 2015.  
<http://shield.nvidia.com/portable-game-player/>
- [61] NVIDIA Corporation. "Using the Bug Database". Retrieved: February 14, 2015.  
[https://wiki.nvidia.com/nvbugswiki/img\\_auth.php/f/f6/Nvbugs\\_overview.ppt](https://wiki.nvidia.com/nvbugswiki/img_auth.php/f/f6/Nvbugs_overview.ppt)
- [62] NVIDIA Corporation. "Visual Computing Module". Retrieved: March 1, 2015.  
<http://www.nvidia.com/object/visual-computing-module.html>
- [63] NVIDIA Corporation. "Why Gremlins". Retrieved: February 18, 2015.  
[https://wiki.nvidia.com/nvbugswiki/index.php/Gremlin\\_-\\_User\\_Documentation#Why\\_Gremlins.3F](https://wiki.nvidia.com/nvbugswiki/index.php/Gremlin_-_User_Documentation#Why_Gremlins.3F)
- [64] "Post Office Protocol - Version 3". Modified: November 1994. Retrieved: February 08, 2015.  
<http://tools.ietf.org/html/rfc1725.html>
- [65] Python Software Foundation. "5.5. Dictionaries". Retrieved: March 02, 2015.  
<https://docs.python.org/2/tutorial/datastructures.html#dictionaries>
- [66] Python Software Foundation. "imaplib - IMAP4 protocol client". Modified: January 01, 2015.  
<https://docs.python.org/2/library/imaplib.html>

- [67] Python Software Foundation. "logging - Logging facility for Python". Retrieved: February 22, 2015. <https://docs.python.org/2/library/logging.html>
- [68] Python Software Foundation. "logging.handler - Logging handlers". Retrieved: February 22, 2015. <http://docs.python.org/2/library/logging.handlers.html>
- [69] Python Software Foundation. "re - Regular expression operations". Modified: January 01, 2015. <https://docs.python.org/2/library/re.html>
- [70] Python Software Foundation. "smtpd - SMTP Server". Modified: January 01, 2015. <https://docs.python.org/2/library/smtpd.html>
- [71] Python Software Foundation. "smtplib - SMTP protocol client". Modified: January 01, 2015. <https://docs.python.org/2/library/smtplib.html>
- [72] Python Software Foundation. "time - Time access and conversions". Retrieved: February 23, 2015. <https://docs.python.org/2/library/time.html>
- [73] Raymon, Eric S. "daemon". Retrieved: March 03, 2015. <http://catb.org/~esr/jargon/html/D/daemon.html>
- [74] Red Hat, Inc., and others. "Overcoming frustration: Correctly using unicode in python2". Retrieved: February 15, 2015. <https://pythonhosted.org/kitchen/unicode-frustrations.html>
- [75] Refsnes Data. "SOAP Syntax". Retrieved: February 15, 2015. [http://www.w3schools.com/webservices/ws\\_soap\\_syntax.asp](http://www.w3schools.com/webservices/ws_soap_syntax.asp)
- [76] Refsnes Data. "XML Tutorial". Retrieved: February 15, 2015. <http://www.w3schools.com/xml/>
- [77] Reitz, Kenneth. "Advanced Usage." Advanced Usage — Requests 2.5.3 Documentation. N.p., n.d. Retrieved: February 24, 2015. <http://docs.python-requests.org/en/latest/user/advanced/#session-objects>
- [78] Reitz, Kenneth. "Requests: HTTP for Humans". Retrieved: February 24, 2015. <http://docs.python-requests.org/en/latest/>
- [79] Reitz, Kenneth. "HTML Scraping". Modified: 2014. Retrieved: February 17, 2015. <http://docs.python-guide.org/en/latest/scenarios/scrape/>
- [80] "Requirements for Internet Hosts -- Application and Support". Modified: October 1989. Retrieved: February 08, 2015. <http://www.rfc-editor.org/rfc/rfc1123.txt>
- [81] Ritger, Andy. NVIDIA Corporation. "NVIDIA ChromeOS Bugs Custom View". Retrieved: March 2, 2015. <https://nvbugswb.nvidia.com/NVBugs5/?dvid=1#75568>

- [82] “run-one package in Ubuntu”. Modified: 2015. Retrieved: February 17, 2015.  
<https://launchpad.net/ubuntu/+source/run-one>
- [83] Saive, Ravi. “CentOS 6.5 Released - Upgrade from CentOS 6.x to CentOS 6.5” Modified: December 2, 2013. Retrieved: February 25, 2015. <http://www.tecmint.com/upgrade-centos/>
- [84] Sajip, Vinay. “Logging HOWTO”. Retrieved: February 22, 2015.  
<https://docs.python.org/2/howto/logging.html>
- [85] Sneddon, Joey-Elijah. “New HP Chromebook 14 Offer Nvidia Tegra K1 Power for \$300”. Modified: September 04, 2014. Retrieved: March 02, 2015. <http://www.omgchrome.com/hp-chromebook-2-tegra-k1-processor/>
- [86] Stallings, William. “Protocol Basics: Secure Shell Protocol”. Retrieved: February 22, 2015.  
[http://www.cisco.com/web/about/ac123/ac147/archived\\_issues/ipj\\_12-4/124\\_ssh.html](http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_12-4/124_ssh.html)
- [87] “Structured Text”. Retrieved: February 17, 2015. <http://www.w3.org/MarkUp/1995-archive/Text.html>
- [88] Tesla. “Tesla Motors”. Retrieved: March 1, 2015. <http://www.teslamotors.com/>
- [89] Thompson, Henry, and Chris Lilley. "XML Media Types." (2014).
- [90] Unicode Table. “Unicode character table”. Retrieved: February 15, 2015. <http://unicode-table.com/en/>
- [91] Unicode, Inc. “What is Unicode?” Modified: January 16, 2015. Retrieved: February 15, 2015.  
<http://unicode.org/standard/WhatIsUnicode.html>
- [92] Unixgeeks.org. “Newbie Introduction to cron”. Retrieved: March 03, 2015.  
<http://www.unixgeeks.org/security/newbie/unix/cron-1.html>
- [93] w3schools. “HTML title Attribute”. Retrieved: February 15, 2015.  
[http://www.w3schools.com/tags/att\\_global\\_title.asp](http://www.w3schools.com/tags/att_global_title.asp)
- [94] Wallen, Jack. “Linux 101: Introduction to sudo”. Modified: May 12, 2010. Retrieved: February 24, 2015. <http://www.linux.com/learn/tutorials/306766:linux-101-introduction-to-sudo>
- [95] WebReference. “What is RSS? (And Atom?)”. Retrieved: March 2, 2015.  
<http://www.webreference.fr/defintions/rss-atom-xml>
- [96] World Wide Web Consortium. "Extensible markup language (XML) 1.1." (2006).
- [97] YAML.org. “GET STARTED”. Retrieved: February 25, 2015. <http://www.yaml.org/start.html>

## 8 Appendices

### Appendix A Web Service Definition Language Example

An example of a WSDL document. This WSDL is for a service for retrieving food recipes [12].

```
<description xmlns="http://www.w3.org/2006/01/wsdl"
             targetNamespace="http://www.brics.dk/ixwt/recipes/wsdl"
             xmlns:x="http://www.brics.dk/ixwt/recipes/wsdl">
  <types>
    <xs:schema
      xmlns:xs="http://www.w3.org/2001/XMLSchema"
      targetNamespace="http://www.brics.dk/ixwt/recipes/wsdl/types"
      xmlns:t="http://www.brics.dk/ixwt/recipes/wsdl/types">

      <xs:import namespace="http://www.brics.dk/ixwt/recipes"
        schemaLocation="recipes.xsd"/>

      <xs:element name="lock">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:length value="16"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:element>

      <xs:element name="lockError" type="xs:string"/>

      <xs:element name="getRecipes">
        <xs:complexType><xs:sequence/></xs:complexType>
      </xs:element>

      <xs:element name="lockRecipe">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="id" type="xs:NMTOKEN"/>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:schema>
  </types>
</description>
```

```

        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="lockRecipeResponse">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="t:lock"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="writeRecipe">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="t:recipe"/>
            <xs:element ref="t:lock"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

<xs:element name="unlockRecipe">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="t:lock"/>
        </xs:sequence>
    </xs:complexType>
</xs:element>

</xs:schema>
</types>

<interface name="recipeserverInterface"
    xmlns:t="http://www.brics.dk/ixwt/recipes/wsdl/types"
    styleDefault="http://www.w3.org/2006/01/wsdl/style/rpc">

```

```

<fault name="lockFault" element="t:lockError"/>

<operation name="getRecipesOperation"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="t:getRecipes"/>
    <output messageLabel="Out" element="t:collection"/>
</operation>

<operation name="lockRecipeOperation"
    pattern="http://www.w3.org/2006/01/wsdl/in-out">
    <input messageLabel="In" element="t:lockRecipe"/>
    <output messageLabel="Out" element="t:lockRecipeResponse"/>
    <outfault ref="x:lockFault" messageLabel="Out"/>
</operation>

<operation name="writeRecipeOperation"
    pattern="http://www.w3.org/2006/01/wsdl/robust-in-only">
    <input messageLabel="In" element="t:writeRecipe"/>
    <outfault ref="x:lockFault"/>
</operation>

<operation name="unlockRecipeOperation"
    pattern="http://www.w3.org/2006/01/wsdl/in-only">
    <input messageLabel="In" element="t:lock"/>
</operation>

</interface>

<binding name="recipeserverSOAPBinding"
    interface="x:recipeserverInterface"
    type="http://www.w3.org/2006/01/wsdl/soap12"
    xmlns:ws="http://www.w3.org/2006/01/wsdl/soap12"
    ws:protocol="http://www.w3.org/2003/05/soap/bindings/HTTP"
    ws:mepDefault=
        "http://www.w3.org/2003/05/soap/mep/request-response"
    xmlns:soap="http://www.w3.org/2003/05/soap-envelope">

```

```
<operation ref="x:getRecipesOperation"/>
<operation ref="x:lockRecipeOperation"/>
<operation ref="x:writeRecipeOperation"/>
<operation ref="x:unlockRecipeOperation"/>
<fault ref="x:lockFault" ws:code="soap:Sender"/>
</binding>

<service name="recipeserver" interface="x:recipeserverInterface">
  <endpoint name="recipeserverEndpoint"
    binding="x:recipeserverSOAPBinding"
    address="http://www.widget.inc/personal/jdoe/recipeserver"/>
</service>
</description>
```

## Appendix B Example of NVBugs Embedded Charts

There are different charts embedded on the online User Interface for NVBugs. The chart in Figure 10 shows the bugs available within a module grouped by the *Priority* field.

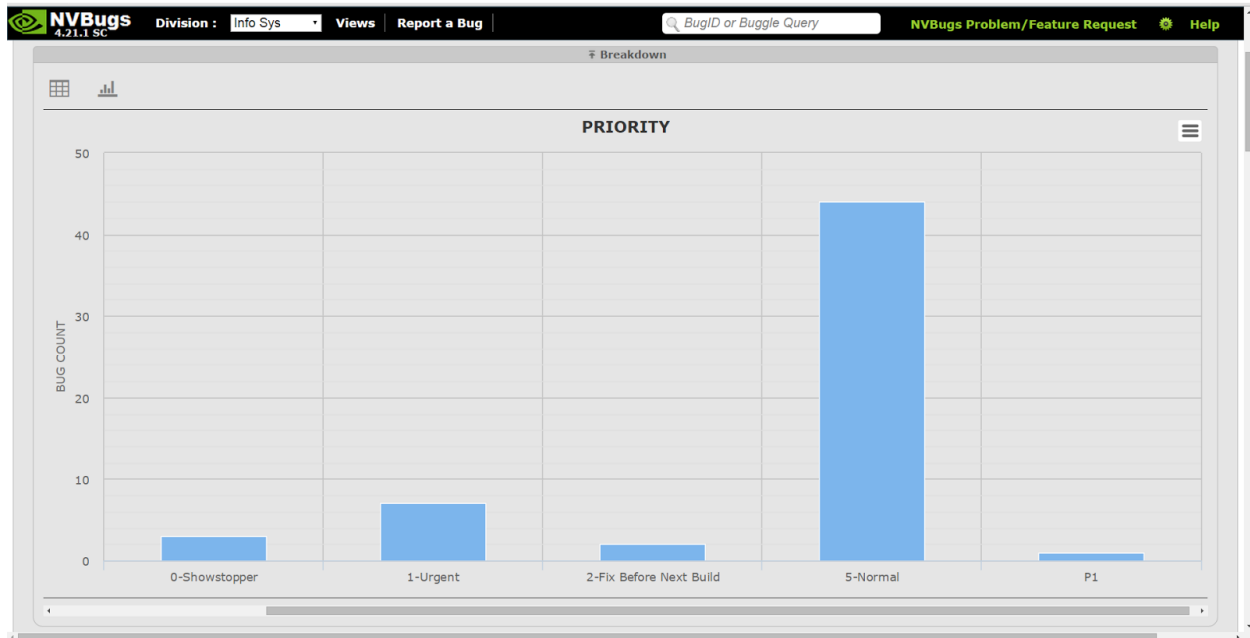


Figure 10: Embedded Charts on NVBugs  
Reference: [47]

## Appendix C Information Retrieved from a Google Bug

Key:

<i>Field Name</i>	Field Type	Field Description
-------------------	------------	-------------------

<i>ID</i>	String	Google Bug ID
<i>ModifiedTimestamp</i>	String	Time in seconds
<i>Modified</i>	String	Datetime string
<i>Status</i>	String	Status ex: "Fixed"
<i>Pri</i>	String	Priority
<i>Proj</i>	String	Project
<i>M</i>	String	Milestone
<i>Summary</i>	String	Summary / Title
<i>Owner</i>	String	Bug owner email
<i>Cr</i>	String	Project Area
<i>Comments</i>	Array	Array of Comments

<i>Comment</i>	String	Comment text
<i>CommentNum</i>	String	Comment number
<i>Date</i>	String	Timestamp
<i>Author</i>	String	Author email
<i>Updates</i>	Array	Updates to fields
	<i>Update</i>	Tuple(String, String) Ex: ("Pri", "2")

## Appendix D Example WSBug XML Object

```
(WSBug) {
  RccaTestEscape =
    (TestEscapePoco) {
      TestEscapeID = 504201
      BugID = 1596371
      TestEscapeType_LookupID = 6
      TestCaseNumber = None
      RccaDaysOpen = None
      IsActive = True
      LastChangedToTbd = None
      BypassTestCaseRequirement = False
      IsDevTestReminderSent = False
      OwnerUserID = None
      RccaInvalidIssue_LookupID = None
      RccaCovEscArea_LookupID = None
      RccaCovEscType_LookupID = None
      DevOwnerID = None
      RootCauseTypeID = None
      DivisionTypeID = 1
      TestEscapeType =
        (TestEscapeType) {
          LookupID = 6
          Name = "Unknown"
          SortRank = 50
          IsActive = False
          IsEditable = False
          IsDeletable = False
          IsEnabled = False
        }
      RCCADivisionType = "SW - QA"
    }
  CustomerExtendedProperty =
    (WSCustomerExtendedProperty) {
      CustomerStatus = None
      SiteTypeID = 0
    }
}
```

```

        CustomerProjectIDList = None
        IsCustomerVisible = False
        CustomerPriorityID = 0
        CustomerOEMInformationID = 0
        CustomerAffectedName = None
        CustomerRequesterUserID = 0
        CustomerSeverityID = 0
        BugID = 1596371
    }
    EffortTrackingDetail =
        (WSEffortTrackingDetail){
            BugID = 1596371
            EffortRemaining = None
            ConfidenceEffortRemainingPercent = None
            TotalEstimatedEffort = None
            PercentComplete = None
            AssignedEngineerWorkTimePercent = None
            CumulativeEffortSpent = 0.0
            AdditionalEffortSpent = None
        }
    IsReadOnly = False
    IsLoadedFromDB = True
    ManDays = 0.0
    RequestDate = "1/7/2015 3:53:56 PM"
    RequestUtcDate = "1/7/2015 11:53:56 PM"
    FixNeededInProductionDate = "1/1/0001 12:00:00 AM"
    Module = "yytest"
    ApplicationDivision = "Software"
    DuplicateBugs = ""
    DuplicateBugList = None
    Comments =
        (ArrayOfWSComment){
            WSComment[] =
                (WSComment){
                    CommentTypeID = 4
                    IsPrivate = True

```

```

        CommentDate = 2015-01-09 15:00:01
        Comment = "This is a comment test"
        AuthorFullName = "Joseph Spicola"
        CommentNumber = 1
    },
}

RequesterFullName = "Joseph Spicola"
RequesterUserName = "28857"
RequesterUserNTAccount = "JSPICOLA"
BugType = "Software"
Version = "version 0.0.1"
VersionFixedAfter = None
FixNeededInBranch = None
OperatingSystems =
    (ArrayOfString){
        string[] =
            "Chrome",
    }
OperatingSystemList = "Chrome"
GTLTaskIDs = ""
GTLTaskIDList = None
PerforceCheckinNumber = None
ARBFullName = None
ARBNTAccount = None
Disposition = "Open issue"
BugAction = "Dev - Open - To fix"
QAEngineerFullName = "Vijendra Patil"
QAEngineerUserName = "14900"
EngineerFullName = "Pallavi Kulkarni"
EngineerUserName = "14041"
DevTechFullName = None
DevTechUserID = 0
Priority = "Unprioritized"
Severity = "7-Task Tracking"
GeographicOrigin = "CA-Santa Clara"
Origin = "App Developer"

```

```

Application = None
ModifiedDate = 2015-01-14 10:11:13.000373
ModifiedUtcDate = 2015-01-14 18:11:13.372032
ProjectedFixDate = "1/1/0001 12:00:00 AM"
ClosedDate = "1/1/0001 12:00:00 AM"
FixedDate = "1/1/0001 12:00:00 AM"
Description = "Test bug to attempt manually editing bug."
DescriptionPlainTextReadOnly = "Test bug to attempt manually editing
bug."

Synopsis = "Test bug for Bug Synchronization Team"
BugID = 1596371
CustomerName = None
CustomerAdditionalList = None
CCFullName = None
IsSendNotification = False
CustomKeywords = "OEMQA:_04_NOT_Tracked_by_OEMQA"
PersonDaysSaved = 0.0
BangBuckFinalRatio = 0.0
Categories = None
BusinessUnits = "MCP/Notebook"
ECOInfo = ""
HardwareOccurrence = ""
}

```

## Appendix E Bug Synchronization Architecture Overview

The flow diagram of our Bug Synchronization service is provided in Figure 11.

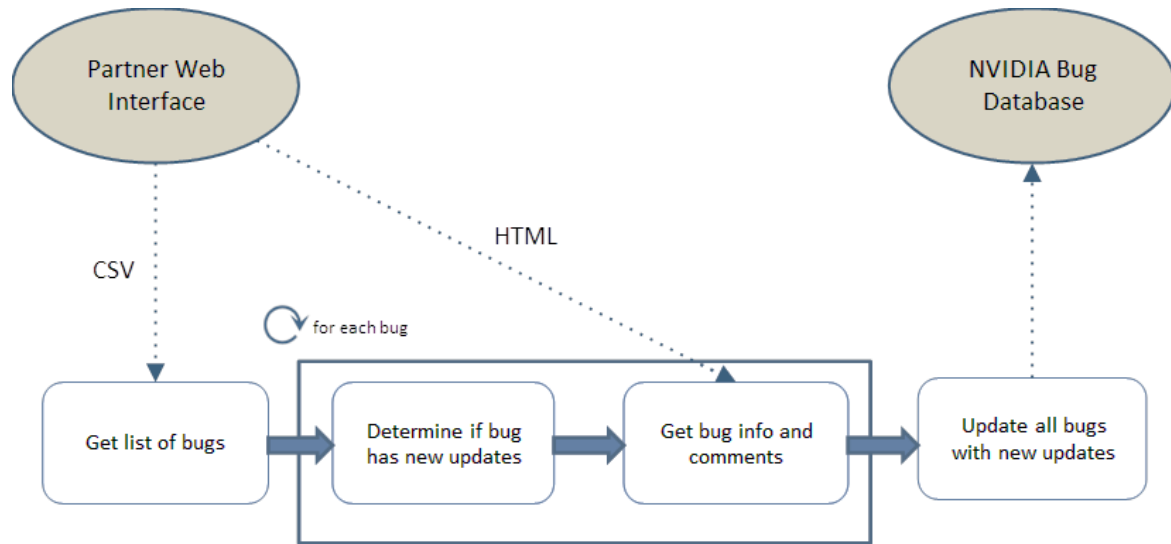


Figure 11: Bug Synchronization Service Flow Diagram

- We start by obtaining a SOAP client to interact with NVBugs.
- We query Google to obtain CSV files containing bugs that are owned by NVIDIA. We do this for both open and closed bugs. We may need to make multiple requests for CSV files depending on the number of bugs – Google truncates each CSV at 100 bugs.
- From the CSV files, we generate a Pythonic dictionary for each Google bug. This dictionary has a number of different fields (Appendix C). The field of immediate importance is the *ModifiedTimestamp* field. An example value in this field is *google-timestamp*.
- For each bug, we use the Google Bug ID to make a request to the *GetSyncAudit* method of the NVBugs Partner Integration API. This call returns the timestamp at which the NVIDIA bug with the input Google Bug ID as its *CustomerBugID* was last synced with the Google Bug. If an NVIDIA bug with the input Google Bug ID as its *CustomerBugID* does not exist, we return a timestamp far enough in the past to ensure that all updates from the Google side will be synced (we currently return the timestamp '1970-01-01 00:00:00.000000'). An example timestamp that would be returned is *nv-timestamp*.

- For each bug, we compare the *google-timestamp* and the *nv-timestamp* to see if there are updates on the Google side that need to be synced. Any changes made between the *google-timestamp* and *nv-timestamp* will need to be synced.
- For each bug, we parse the Google bug's HTML to get all the updates that need to be synced, and store them in a Pythonic array of dictionaries. This array is called *bugs\_to\_sync*.
- We then iterate over *bugs\_to\_sync*.
- For each bug, we generate the required XML to sync the pending updates, and send the XML as a parameter to the *SavePartnerBug* method of the NVBugs Partner Integration API.
- For each bug, we parse the return values of the *SavePartnerBug* function, and log appropriate success or failure messages.

## Appendix F Configuration File

"""Config file to indicate which NVBugs server to use for bug-synchronization.

Possible values are 'DEV' or 'PROD'.

PLEASE ENSURE YOU USE EITHER ONE OF THESE VALUES EXACTLY.

The DEV server can be used to test and debug the bug-synchronization service.

The PROD server should be used to deploy the service.

KEEP IN MIND THAT THE SERVICE WILL MODIFY PRODUCTION BUGS WHEN SET TO PROD.

PLEASE BE EXTREMELY CAREFUL WHEN SETTING THE SERVER TO PROD - MAKE SURE THE SERVICE IS CONFIGURED IN A STATE WHERE YOU CAN RUN AND FORGET IT.

Configures variables for the bug-synchronization service.

Authors: jspicola, kbrann, tnarayan

"""

#set SERVER to either 'PROD' or 'DEV'

SERVER='DEV'

#####

# YOU SHOULD NOT NEED TO MODIFY ANYTHING BELOW THIS HEADER.

#####

##### Deployment configuration specific constants

if SERVER == 'PROD':

    NVIDIA\_USERNAME = 'SvcGoogleChromeSync'

    NVIDIA\_ENCRYPTED\_PASSWORD\_FILE = (

        '/home/kbrann/bug-sync/encrypted\_passwords/nvidia\_encrypted\_pass-prod.txt')

    NVBUGS\_PARTNER\_API\_URL =

    'https://nvbugswb.nvidia.com/NVBugsWebService/PartnerIntegration.asmx'

elif SERVER == 'DEV':

    NVIDIA\_USERNAME = 'SvcGoogleChromeSyncDev'

    NVIDIA\_ENCRYPTED\_PASSWORD\_FILE = (

```

        '/home/kbrann/bug-sync/encrypted_passwords/nvidia_encrypted_pass-
dev.txt')
    NVBUGS_PARTNER_API_URL =
'http://hqdvbugswb03.nvidia.com/NVBugsWebService/PartnerIntegration.asmx'

else:
    print 'Please set the SERVER variable in config.py to either PROD or
DEV.'
    print 'No other values supported.'
    exit(-1)

GOOGLE_EMAIL = 'Bug-sync@nvidia.com'
GOOGLE_ENCRYPTED_PASSWORD_FILE = (
    '/home/kbrann/bug-sync/encrypted_passwords/google_encrypted_pass.txt')

NVBUGS_PARTNER_API_WSDL_URL = '%s?WSDL' % NVBUGS_PARTNER_API_URL

##### General constants
# constants for the Google authentication pages
GOOGLE_LOGIN_URL = "https://accounts.google.com/ServiceLogin"
GOOGLE_AUTH_URL = "https://accounts.google.com/ServiceLoginAuth"

# URL to retrieve html info page on bug in chrome-os-partner project
GOOGLE_BUG_URL = 'https://code.google.com/p/chrome-os-
partner/issues/detail?id='

# URL to retrieve CSV of all bugs in chrome-os-partner project
GOOGLE_CSV_URL = 'https://code.google.com/p/chrome-os-
partner/issues/csv?can=1&'

```

## Appendix G Deployment Instructions for Google Chrome Sync

### Deployment Instructions for Google Chrome Bug-Sync

(02/20/2015)

#### 1. Dependencies

The Google Chrome Bug-Sync service is written in Python - Version 2.7.  
This version of Python must be installed on the machine running the service.  
You can check the python version by running 'python --version'.

The following modules are used in the Google Chrome Bug-Sync service and must be installed in the Python - Version 2.7 lib directory (/lib/python2.7):

- bs4 (BeautifulSoup - <http://www.crummy.com/software/BeautifulSoup/>)
- requests (<http://docs.python-requests.org/en/latest/>)
- yaml (<http://pyyaml.org/>)
- suds (<https://pypi.python.org/pypi/suds>)
- six (<https://pypi.python.org/pypi/six>)
- dateutil (<https://pypi.python.org/pypi/python-dateutil>)

#### **TO GET THIS SETUP IN PROD:**

Copy the directory Python-2.7.6 from /usr/bin/ on the hqdvbugs01 (dev) VM to /usr/bin/ on the hqncbugs01 (prod) VM. This directory will contain the necessary modules.

2. Copy the directory GoogleChromeSync from /usr/local/ on the hqdvbugs01 VM to /usr/local/ on the hqncbugs01 VM.  
(The project can also be found at: <http://git-master/r/#/admin/projects/chromeos/bug-sync>)

Change directory to /GoogleChromeSync/bug-sync and enter the command 'tree'.

The following files and directories should be present (as of 2/23/2015):

```
|-- Documentation
| |-- README.txt
| |-- Security
| |-- encrypting_password.py
|-- config.py
|-- crontab.txt
|-- encrypted_passwords
| |-- google_encrypted_pass.txt
| |-- nvidia_encrypted_pass-dev.txt
|-- get_nv_client.py
|-- google_auth.py
```

```

|-- google_partner_sync.py
|-- html_escape_entities.py
|-- logging.yaml
|-- login_info.py
|-- logs
|   |-- critical.log
|   |-- errors.log
|   |-- info.log
|   |-- verbose.log
|   `-- warning.log
|-- partner_db_interaction.py
|-- setup_logging.py
|-- stats
|   |-- excel_objects
|   |   `-- frontend.txt
|   `-- modules
|       |-- checkbox.txt
|       |-- getallbugsfuctions.txt
|       |-- getengineerfunctions.txt
|       |-- getmodulefunctions.txt
|       |-- getprojectfunctions.txt
|       |-- gettrendfunctions.txt
|       `-- stringgeneration.txt
|-- sync_exceptions.py
`-- unicode_converter.py

```

The *stats* directory is just a container for files relating to a stretch goal of the synchronization service. None of the files and directories within the *stats* directory are relevant to the actual synchronization process itself - they can be safely ignored.

### 3. Encrypt/supply passwords:

Our passwords are encrypted using GPG. The `login_info.py` file includes functions to decrypt the passwords from the encrypted password files specified in `config.py`.

*Instructions to encrypt passwords using GPG are in the file  
'Security/EncryptingGPGPasswords.txt'.*

*(You **must** either generate the encrypted password files using GPG encryption or supply the passwords in a different way, for this process to run.)*

The generated password files must be put within the “encrypted\_passwords” folder. The following convention is expected for filenames:

- `google_encrypted_pass.txt` is the file containing the encrypted password for the Google account with username “bug-sync@nvidia.com”.

- `nvidia_encrypted_pass-dev.txt` is the file containing the encrypted password for the NVIDIA service account that has access to the development version of NVBugs. Currently, this is configured to be the “SvcGoogleChromeSyncDev” service account.
  - `nvidia_encrypted_pass-prod.txt` is the file containing the encrypted password for the NVIDIA service account that has access to the production version of NVBugs. Currently, this is configured to be the “SvcGoogleChromeSync” service account. We do not have the password for this account, hence there is no encrypted password file included by default. **THIS FILE MUST BE GENERATED AND PUT IN THIS LOCATION WHEN MOVING FROM DEV TO PROD.**
  - If you choose to hardcode the passwords, modify the following functions in `login_info.py`:
    - `get_google_login_password()` and `get_nvidia_login_password()`: give them the valid return statement.
    - `get_decrypted_password()` function: deprecate this, or delete it if you never plan on using GPG encryption in the current manner.
4. In `config.py`, the following constants should be changed:
- `SERVER`  
Either PROD or DEV. No other values will be accepted.

You should not need to modify the following two values unless your filepaths are different from the default “local” directory’s filepaths, or if the passwords are changed:

- `NVIDIA_ENCRYPTED_PASSWORD_FILE`  
The full path to the encrypted file for the `NVIDIA_USERNAME` account’s password. The password can be encrypted using the process described at the bottom of this file. Note that if your filepaths are different, the location will need to change in two places: line 27 and line 33.
- `GOOGLE_ENCRYPTED_PASSWORD_FILE`  
The full path to the encrypted file for the `GOOGLE_EMAIL` account’s password. The password can be encrypted using the process described at the bottom of this file.

The other constants in the file are:

- `NVIDIA_USERNAME`  
This is the name of the service account that syncs the bugs. Configured to the DEV or PROD service account depending on the value of the `SERVER` variable.
- `NVBUGS_PARTNER_API_URL`  
This is the link to the NVBugs partner integration API. Set to the DEV or PROD version of the NVBugs Partner API depending on the value of the `SERVER` variable. Note that the API must have the `GetSyncAudit` method added to it (this method was added specifically

for one-way synchronizations, and hadn't been pushed up to the production API as of 02/20/2015 3:20PM).

Edit on 02/24/2015 9:40AM

The *GetSyncAudit* method is now available on the production NVBugs Partner Integration API.

f. **GOOGLE\_EMAIL**

The NVIDIA email account that has access to the Google *chrome-os-partner* project on Google Project Hosting. Set to bug-sync@nvidia.com. Should not need to change unless we want to sync using a different account.

The other constants in the file should not need to be changed for deployment. They require changing only if Google changes the URL at which the partner project is hosted.

5. Edit the crontab.txt file to contain the correct file paths.

**TO GET THIS SETUP ON PROD:**

The crontab.txt file will already contain a properly formed crontab that was used on dev. This crontab.txt file should require no changes and will look like this:

```
*/10 * * * * /local/GoogleChromeSync/bug-sync/run-one-1.17/run-one /bin/Python-2.7.6/python /local/GoogleChromeSync/bug-sync/google_partner_sync.py >/dev/null 2>&1
```

*In the case that you would like to create a new crontab.txt file, follow these instructions:*

Crontab should be constructed as follows:

```
* /5 * * * * A B C >> D 2>&1
```

where A,B,C,D are as follows:

A: This is the full file path to the run-one executable.

Using the run-one command ensures that the crontab will not run the script if the last run has not completed yet.

Change to correct file path for run-one in your directory setup.

B: This is the full file path to Python - Version 2.7.

Change to the correct file path for Python - Version 2.7 in your directory setup.

C: This is the full file path to our main function script, google\_partner\_sync.py.

Change to the correct file path to google\_partner\_sync. in your directory setup.

D: This is the full file path to the a text file to direct stdout output to.

Change to the correct file path in your directory setup.

```
*/5 * * * * : Defines the sync interval- do not change.
>> : This indicates to redirect program stdout output- do not change
2>&1 : This is necessary for output redirection- do not change.
```

Example of a complete crontab statement:

```
* /5 * * * * /home/kbrann/bug-sync/run-one-1.17/run-one /home/kbrann/bin/python2.7  
/home/kbrann/bug-sync/google_partner_sync.py >> /home/kbrann/bug-sync/output.txt 2>&1
```

6. Run command 'crontab crontab.txt' to load the crontab.

*This will overwrite any existing cron configuration. If you do not want to lose your current cron configuration, run the command 'crontab -e' to edit your crontab and add the line from the 'crontab.txt' file at the end of the crontab.*

Once this step is complete the script google\_partner\_sync.py should run every 10 minutes. Information will be logged to log files in the logs/ directory. All stdout and stderr output will be directed to the file specified in the crontab statement.

## Instructions for encrypting passwords with GPG

(02/20/2015)

THESE INSTRUCTIONS ARE FOR LINUX MACHINES THAT DO NOT COME BUNDLED WITH GPG ALREADY INSTALLED. FOR MACHINES THAT COME PREINSTALLED WITH GPG (SUCH AS CENTOS), SKIP THE 'INSTALL GPG' STEPS.

- You can check if gpg is installed by running the simple "gpg --list-keys" command.
  - If it gives you an error, then try installing gpg.
  - Otherwise, skip installation step.

LOCAL MACHINE: (The machine running our service)

- Install GPG
  - (for ubuntu) sudo apt-get install gpg
- Generate a secret key
  - gpg --gen-key
    - Follow instructions, giving it a relevant name
      - Example: "example\_name@nvidia.com"
      - GPG will ask you to develop entropy. During this time, GPG will use your keystrokes/mouse actions/etc to generate different values for your secret keyring. You are free to generate the necessary how you want; check your email, generate a second key in another window, even mash the keyboard continuously until GPG generates enough entropy.
    - (Optional): Generate a revocation certificate
      - This will enable you to revoke the key

- If you publish the revoked key, any user attempting to use your revoked public key will be notified of its revoked status.
- Export the public key to a text file
  - `gpg --export -a "example_name@nvidia.com" > public.key`

ADMIN MACHINE: (The machine used by admin to generate passwords for Local Machine)

- Install GPG
  - (for ubuntu) `sudo apt-get install gpg`
- Generate a secret key
  - Follow instructions, giving it a relevant name
    - Example: "bug\_admin@nvidia.com"
    - GPG will ask you to develop entropy. During this time, GPG will use your keystrokes/mouse actions/etc to generate different values for your secret keyring. You are free to generate the necessary how you want; check your email, generate a second key in another window, even mash the keyboard continuously until GPG generates enough entropy.
  - (Optional): Generate a revocation certificate
    - This will enable you to revoke the key
      - If you publish the revoked key, any user attempting to use your revoked public key will be notified of its revoked status.
- Import Local Machine's key
  - `gpg --import public.key`
    - Must have copied over the public.key file from the Local Machine
- Export Admin Machine's key to a text file
  - `gpg --export -a "bug_admin@nvidia.com" > admin_public.key`

LOCAL MACHINE:

- Import Admin Machine's key
  - `gpg --import admin_public.key`
    - Must have copied over the admin\_public.key file from the Admin Machine

The machines are now ready to encrypt and decrypt messages for each other.

If you want to double check, on each machine, if you use the command:

- `gpg --list-keys`  
You will see both the admin and the local machine's public keys.

If you use the command:

- `gpg --list-secret-keys`  
You will only see the private key for the machine that you are on.

## Appendix H Documentation

This is the README for the NVIDIA ChromeOS team bug-sync intern project.

### ----- PURPOSE -----

The purpose of this project is to provide a service that can synchronize Google Partner Bugs to NVBugs.

Google Partner Bugs is the name used to refer to the bugs for the chrome-os-partner project hosted on Google Project Hosting. The project can be found at

<https://code.google.com/p/chrome-os-partner/>

NVBugs is NVIDIA's internal bug tracker. The production server for NVBugs can be found at

<https://nvbugswb.nvidia.com/NVBugs/default.aspx>.

This system is intended to be run as a standalone process via cron. The main function for controlling the system is contained in the google\_partner\_sync.py file.

### ----- SETUP -----

The source code for this service is checked into Gerrit. The project can be found at

<http://git-master/r/#/admin/projects/chromeos/bug-sync>

For the correct version of Git, use /bin/git-1.8.2.3/git.

Clone the project into a local "bug-sync" directory.

Ensure all Dependencies (listed below) are installed on the system.

Change working directory to be the "bug-sync" directory.

To sync all bugs with new updates on the Google side once:

- Run the main script:

```
python google_partner_sync.py
```

- All bugs with unsynced updates present on the Google side will be synced with NVBugs. This includes creating new bugs and copying over comments and field changes for existing bugs.
- The service will output the results of the synchronization to the file "info.log" in the "logs" folder.
- Any sync failures will be noted in the output.  
Most errors are self-fixing on the next run of the script. Critical errors (which the script cannot recover from) include changed authentication credentials for either the Google account or the NVIDIA service account.

To run the service in the background and periodically sync updates:

- Open the "crontab.txt" file in a text editor.
- Ensure that the cron job settings are configured as required. For more information on configuring cron job timings, you can refer to

<http://content.hccfl.edu/pollock/unix/crontab.htm>

The default "crontab.txt" file has the cron job configured to run every 10 minutes.

- The cron job needs to contain "run-one" after the variables have been set. This ensures that only a single process of the service is run at

any time. If the service takes longer than the time the cronjob waits, it will silently kill itself, and won't attempt to run until the next scheduled time.

The "run-one" module is a Ubuntu Debian application that has been configured to work on the CentOS VM and is located in the local bug-sync directory. An explanation of "run-one" can be found at: <http://blog.dustinkirkland.com/2011/02/introducing-run-one-and-run-this-one.html>

To download "run-one", visit:  
<https://launchpad.net/ubuntu/+source/run-one>

- To view your current crontab settings, run:

```
crontab -l
```

- If you already have a crontab, run:

```
crontab -e
```

Add the line from the "crontab.txt" file to the end of the crontab file.

- Else, run:

```
crontab crontab.txt
```

This will load "crontab.txt" as the crontab job.

- View your crontab again:

```
crontab -l
```

Ensure that the line from the "crontab.txt" file is present in the output.

- The service will run periodically as configured. The output will be captured in the files within the "logs" folder. Each file in the folder corresponds to the severity level of the log message. The following levels are used:

- VERBOSE
- INFO
- WARNING
- ERROR
- CRITICAL

Severity levels stack: ERRORS are printed at the error, warning, info, and verbose levels. Thus, the "verbose.log" file will include \*all\* log messages.

Log files are rotated periodically, as per the settings configured in the "logging.yaml" file.

-----

#### APPROACH USED TO SYNC GOOGLE PARTNER BUGS

-----

Google provides a list of open bugs via CSV files. Each CSV file is truncated to hold 100 results. Parsing through all the CSV files thus enables us to get a list of all open Google Partner Bugs. Using the Google Bug ID and the time the bug was last modified on the Google side, we query the NVBugs database to understand if the bug contains updates on the Google side that have not yet been synchronized to the NVIDIA side.

If unsynchronized updates are present, we scrape the HTML for the page that displays the bug on the Google side. This enables us to get all the information about the bug, along with all the comments made on the bug. We then figure out which comments (if any) have not yet been synchronized to the NVIDIA side. At this point, we know what information from the Google side needs to be synced to the NVIDIA side.

We then generate the XML for sending these updates to the NVIDIA side. While generating the XML, we ensure that any fields that cannot map directly between Google and NVIDIA still have their value synced to the NVIDIA side: we use custom NVBugs fields for this purpose. Once the XML is generated, we use a function exposed via the NVBugs Partner Integration API to sync updates to the NVBugs database.

-----  
CODE FILES & DESCRIPTION  
-----

Code files used in the system and their functionality is as follows:

- config.py - contains configurable constants for the service.
- crontab.txt - contains the cron settings for the service.
- Documentation/
  - README.txt - this readme file.
  - Security/
    - encrypting\_password.py - script to generate the encryption keys for the master account (security is setup using GPG, see "SECURITY" below.
- encrypted\_passwords/
  - google\_encrypted\_pass.txt - Google account's encrypted password.
  - nv\_encrypted\_pass-dev.txt - NVIDIA service account's encrypted password.
- get\_nv\_client.py - contains functionality to get a Client object that exposes the functionality of the NVBugs Partner Integration API

- `google_auth.py` - contains functionality to log into the Google and scrape Google resources (CSV/HTML) that require authentication.
- `google_partner_sync.py` - contains main function.
- `html_escape_entities.py` - contains utility function to convert special characters in strings to make the strings HTML compliant.
- `logging.yaml` - the configuration info for logging.
- `login_info.py` - contains functions used to securely obtain login credentials.
- `logs/` - the folder to contain the log files
- `partner_db_interaction.py` - contains functions used to interact with the NVBugs database, including querying when a bug was last updated and saving updates.
- `setup_logging.py` - contains the function used to read the logging config info.
- `stats/` - the folder to contain unrelated code for a bug statistics project.
- `sync_exceptions.py` - contains class used to capture Authentication Errors.
- `unicode_converter.py` - contains utility functions to convert strings to Unicode.

## ----- SECURITY -----

Passwords used by the accounts the service accesses are encrypted using gpg.

The local machine running the service generates a secret key using gpg. An admin account elsewhere generates a secret key.

Both accounts generate a public key, and can publish or share them. The admin and local machine must sign each other's public keys.

The admin then encrypts the relevant password using the script found in /Security/encrypting\_password.py, using the local machine's public key. The local machine running the service will need to import the file to the /encrypted\_passwords folder, where it will be decrypted using login\_info.py by calling gpg's decrypt function.

This process ensures that only the local machine will have the ability to decrypt the password using it's secret key ring.

-/Security/encrypting\_password.py:

This script is intended to be run by the admin account.

In order to use the encrypting\_password script, the machine running the script must have a gpg generated secret-key. In order for the script to generate an encryption usable by the local machine, the admin account running it needs to have it's public key signed by the local machine.

The admin starts the script, enters the following:

- Password that needs to be encrypted
- Email account of the owner of the keyring the password is intended for
- Password for the admin's secret keyring

The script will encrypt the password and print it out to a 'pass\_sec'

file. The 'pass\_sec' file can be transferred to the local machine, and renamed appropriately. The service will now use the encrypted password for the relevant credentials.

---

#### EXTERNAL DEPENDENCIES

---

```
bs4 (google_auth)
requests (google_auth, google_partner_sync)
run-one (crontab)
gpg (login_info)
yaml (setup_logging)
suds (get_nv_client)
six (generic Python dependency)
dateutil (google_auth)
```

---

#### AUTHORS

---

The authors of this codebase and the accompanying documentation are:

```
kbrann@nvidia.com
tnarayan@nvidia.com
jspicola@nvidia.com
```

## Appendix I Bug Synchronization Statistics

Table 3: Bug Synchronization Service Statistics

Bugs Retrieved	Bugs Updated	Time (seconds)
30	1	10.53316402
30	0	7.589803934
30	0	7.315895796
30	0	7.432412863
30	0	8.165171146
30	0	9.914444923
30	3	25.76152301
30	0	6.547803879
30	0	7.522241116
30	0	6.963369846
30	0	11.343889
30	0	6.804688931
30	0	7.12361598
30	0	6.537559986
30	0	6.162681818
30	0	7.11659503
30	0	13.93101811
30	0	7.922743082
30	0	8.218853951
30	0	8.078969002
30	0	13.70051789
30	0	9.09929204
30	0	6.9185009
30	0	8.972236872
30	0	7.139943838

30	0	7.460359097
30	0	7.363718987
30	0	7.71273303
30	0	7.9131248
30	0	7.30921483
30	0	7.467134953
30	0	7.012370825
30	0	7.385293007
30	0	7.032801151
30	0	7.196313858
30	0	8.046500206
30	0	6.913458824
30	0	8.216697931
30	0	7.021950006
30	0	7.965540886
30	0	6.284950972
30	0	6.973535061
30	0	6.868589163
30	0	6.99032402
30	0	6.946500063
30	0	7.732135057
29	0	8.430742025
29	0	7.049008846
29	0	8.940618038
29	0	10.3945291
29	0	7.766481876
29	0	8.429244041
29	0	8.26897788
29	0	7.317604065

29	0	7.689349175
29	0	7.755438089
29	0	6.768245935
29	0	7.52431798
29	0	17.46087885
29	0	7.622886896
29	0	7.867061138
29	0	7.189815044
29	0	7.397203922
29	0	8.118759871
29	0	7.818226099
29	0	6.484632015
29	0	7.496836901
29	0	6.4319911
29	0	7.586184978
29	0	9.651571989
29	0	7.615714073
29	0	6.904597044
29	0	7.671758175
29	0	6.84355402
29	0	8.172986984
29	0	7.208384037
29	0	7.297168016
29	0	22.40175509
29	0	7.453086853
29	0	7.505218983
29	0	8.745570183
29	0	7.704330921
29	0	7.09620595

29	0	7.086565971
29	0	7.079473972
29	0	6.681159973
29	0	6.993405819
29	0	6.637966871
29	0	7.137184858
29	0	8.768369198
29	0	6.827040911
29	0	7.287039995
29	0	7.035991907
29	0	7.110824823
29	0	7.154912949
29	0	7.124298096
29	0	7.922944069
29	0	7.763846159
29	0	8.232103109
29	0	6.48289299